

Numero 20
Ottobre
2015

ArsT_EXnica

Rivista italiana di T_EX e L^AT_EX

GU_{IT}

<http://www.guitex.org/arstexnica>



G_UIT – Gruppo Utilizzatori Italiani di T_EX

ArsT_EXnica è la pubblicazione ufficiale del G_UIT

Comitato di Redazione

Claudio Beccari – *Direttore*

Roberto Giacomelli – *Comitato scientifico*

Enrico Gregorio – *Comitato scientifico*

Ivan Valbusa – *Comitato scientifico*

Renato Battistin, Riccardo Campana,

Massimo Caschili, Gustavo Cevolani,

Massimiliano Dominici, Tommaso Gordini,

Carlo Marmo, Gianluca Pignalberi,

Ottavio Rizzo, Gianpaolo Ruocco,

Enrico Spinielli, Emiliano Vavassori

ArsT_EXnica è la prima rivista italiana dedicata a T_EX, a L^AT_EX 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 G_UIT, 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 ArsT_EXnica, 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 T_EX e L^AT_EX 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 accettati, 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@guitex.org.

Nota sul Copyright

Il presente documento e il suo contenuto è distribuito con licenza  Creative Commons 2.0 di tipo “Non commerciale, non opere derivate”. È possibile, riprodurre, distribuire, comunicare al pubblico, esporre al pubblico, rappresentare, eseguire o recitare il presente documento alle seguenti condizioni:

- ① **Attribuzione:** devi riconoscere il contributo dell'autore originario.
- ② **Non commerciale:** non puoi usare quest'opera per scopi commerciali.
- ③ **Non opere derivate:** non puoi alterare, trasformare o sviluppare quest'opera.

In occasione di ogni atto di riutilizzazione o distribuzione, devi chiarire agli altri i termini della licenza di quest'opera; se ottieni il permesso dal titolare del diritto d'autore, è possibile rinunciare ad ognuna di queste condizioni.

Per maggiori informazioni:

<http://www.creativecommons.org>

Associarsi a G_UIT

Fornire il tuo contributo a quest'iniziativa come membro, e non solo come semplice utente, è un presupposto fondamentale per aiutare la diffusione di T_EX e L^AT_EX 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 T_EX

c/o Università degli Studi di Napoli Federico II

Dipartimento di Ingegneria Industriale

Via Claudio 21

80125 Napoli – Italia

<http://www.guitex.org>

guit@sssup.it

Redazione ArsT_EXnica:

<http://www.guitex.org/arstexnica/>

arstexnica@guitex.org

Codice ISSN 1828-2369

Stampata in Italia

Napoli: 15 Ottobre 2015

GUI meeting 2015

DODICESIMO CONVEGNO NAZIONALE
SU T_EX, L^AT_EX E TIPOGRAFIA DIGITALE

17 ottobre 2015

Polo scientifico e tecnologico “Fabio Ferrari”,
via Sommarive 9, Povo, Trento
Edificio Povo 2, Aula B107
Università degli Studi di Trento

Programma del convegno

- 9:00 Benvenuto. Inizio dei lavori.
- 9:15 *I calcoli matematici in pdf_{tex}*. Claudio Beccari.
- 9:45 *Il pacchetto ECG*. Ezio Aimé e Marco Scavino.
- 10:15 *Canzonieri online con L^AT_EX*. Francesco Endrici e Matteo Conci.
- 10:45 — Pausa caffè (30 min).
- 11:15 *Creare un database bibliografico da un file di testo. Una proposta*. Matteo Fadini.
- 11:45 *Zbl-build: a GUI interface for Bib_lat_ex*. Guido Milanese.
- 12:15 *MLBib_TTeX 1.4: the New Version*. Jean-Michel Hufflein.
- 12:45 — Pausa pranzo (1 h 45 min).
- 14:30 *Generare documenti L^AT_EX con diversi linguaggi di programmazione*. Roberto Giacomelli e Gianluca Pignalberi.
- 15:00 *Ricordi di un cacciatore di spazi spuri*. Enrico Gregorio.
- 15:30 *Easy T_EX automation with arara*. Paulo Roberto Massa Cereda.
- 16:00 *Animazioni e grafici interattivi in L^AT_EX*. Grazia Messineo e Salvatore Vassallo.
- 16:30 — Pausa caffè (30 min).
- 17:00 Discussione sulle prospettive del gruppo.
- 18:00 Chiusura dei lavori. Arrivederci.

La partecipazione è libera e gratuita, previa registrazione entro il 10 ottobre.

Registrazione online: www.guitex.org/home/meeting

Informazioni: Francesco Endrici, francescoendrici@gmail.com



www.guitex.org



UNIVERSITÀ DEGLI STUDI
DI TRENTO

ArsT_EXnica

Rivista italiana di T_EX e L^AT_EX

Numero 20, Ottobre 2015

Claudio Beccari	
Editoriale	5
Claudio Beccari	
I calcoli matematici in pdftex	7
Ezio Aimé, Marco Scavino	
Il pacchetto ECG	16
Francesco Endrici, Matteo Conci	
Canzonieri online con L ^A T _E X	23
Matteo Fadini	
Creare un database bibliografico da un file di testo. Una proposta . . .	27
Guido Milanese	
Zbl-build: a GUI interface for Biblatex	31
Jean-Michel Huffle	
MIBB _T _E X 1.4: the New Version	35
Roberto Giacomelli, Gianluca Pignalberi	
Generare documenti L ^A T _E X con diversi linguaggi di programmazione .	40
Enrico Gregorio	
Ricordi di un cacciatore di spazi spuri	74
Paulo Roberto Massa Cereda	
Easy T _E X automation with arara	84
Grazia Messineo, Salvatore Vassallo	
Animazioni e grafici interattivi in L ^A T _E X	90

Gruppo Utilizzatori Italiani di T_EX

Editoriale

Claudio Beccari

Questo Meeting 2015 che si svolge a Trento ci riserva un bel numero di interventi che spaziano su una buona varietà di temi. Inoltre, non è la prima volta che in un Meeting succede una cosa del genere, ma abbiamo anche un intervento registrato in un breve filmato che arriva nientemeno che dal Brasile.

Il mio intervento riguarda i calcoli che si possono fare con il motore di composizione *pdf_{tex}*; ne ho già parlato in almeno un altro articolo pubblicato su questa rivista, ma nella visuale della problematica esposta in quell'articolo. Qui cerco di essere più generale indicando come fare i calcoli senza ricorrere al pacchetto *calc* o al pacchetto *fp*, né alla libreria matematica di *pgfplots*, né al linguaggio Lua incorporato in *luatex*, né, infine, alla nuova funzionalità del linguaggio L³ che consente di eseguire i calcoli in notazione *floating point*. Di per sé il motore *tex* e i suoi discendenti non formano una calcolatrice sufficientemente avanzata per fare tutti i calcoli che potrebbero essere necessari e con sufficiente precisione, ma programmando bene e sfruttando le nozioni di base del calcolo numerico si riescono a fare cose piuttosto utili; alcuni esempi lo dimostrano.

Ezio Aimé e Marco Scavino ci presentano un pacchetto molto utile per visualizzare tracciati di elettrocardiogrammi tipici; costruiti perciò a tavolino per mostrare sequenze di onde di elettrocardiogrammi dove sono riportate e ben evidenziate le caratteristiche di tali onde, sia nella loro normalità sia in presenza di andamenti che possono segnalare certe patologie. Il risultato dovrebbe essere particolarmente gradito ai non pochi medici che usano i programmi del sistema T_EX.

Francesco Endrici ci presenta un programma basato sul sistema T_EX, ma non solo, per comporre un canzoniere completo di parole e spartito, utile sia per la liturgia sia nelle attività di gruppi organizzati. Il programma serve anche per mettere in rete questi canzonieri mediante le funzionalità dei linguaggi usati per le pagine Web. I risultati sono molto buoni; l'esperienza che ne deriverà permetterà di eliminare alcuni punti delicati e di aggiungere ulteriori funzionalità.

Matteo Fadini ci presenta una proposta per creare un database bibliografico a partire da un file composto in uno dei soliti word processor. Esistono diversi programmi capaci di convertire un file di testo formattato come quelli prodotti con quei word processor, ma nessuno riesce a prelevare i dati dalla bibliografia di quei testi, di solito composta usando

le impostazioni prescrittive di quei programmi; il procedimento proposto in questo intervento elabora un file di testo composto con i word processor e produce un file *.bib* correttamente composto ed elaborabile con il programma *biber* per ricomporre la bibliografia con il mark-up di L^AT_EX, in modo da poter adattare la composizione a precise esigenze editoriali.

Guido Milanese ci presenta una interfaccia grafica per gli utenti per impostare il pacchetto *bib_{latex}* in modo piuttosto semplice; essa evita ai neofiti di dover consultare le informazioni necessarie, che però sono sparse lungo la documentazione del pacchetto *bib_{latex}*. Per ogni possibile chiave le scelte possibili sono presentate a schermo con una brevissima descrizione; un semplice click permette di scegliere il valore desiderato fra quelli accettabili. Questa interfaccia grafica sarà molto apprezzata da tutti coloro che non amano leggere la documentazione.

Jean-Michel Hufflein ci presenta un ulteriore perfezionamento del suo programma *MIBibTeX* nella versione 1.4. Il programma non è ancora disponibile nelle grandi distribuzioni, ma lo sarà a breve. Le funzionalità sono molto arricchite rispetto a quelle presentate nel Meeting 2014 di Verona. Il programma è stato quasi completamente riscritto per poter usare la codifica di entrata UNICODE e come elaboratore di bibliografie il programma *biber*. Penso che gli utenti di questa prossima nuova versione saranno molto soddisfatti delle nuove funzionalità.

Roberto Giacomelli e Gianluca Pignalberi ci presentano alcuni metodi per generare file sorgente col mark-up L^AT_EX partendo da testi scritti con i linguaggi più disparati e da ricomporre in modo uniforme e corretto seguendo la logica tipografica sottostante a L^AT_EX. Il progetto sviluppato dagli autori è piuttosto complesso vista la varietà dei testi sorgente e l'unitarietà del file *.tex* contenente il testo sorgente per la compilazione con uno dei compilatori basati su L^AT_EX. Gli autori ci presentano problematiche di diverso genere e le soluzioni che hanno sviluppato.

Enrico Gregorio ci stupisce sempre con la sua profonda conoscenza del sistema T_EX. In questo articolo ci racconta la storia delle sue cacce allo spazio spurio. Questi sono quei terribili spazi che talvolta appaiono dove non dovrebbero nei testi composti e che non si riesce a capire da dove arrivino. Di solito nascono dai fine riga dopo una graffa chiusa; sì, ma quale graffa? Altre volte nascono

da situazioni molto più difficili da trovare perché bisognerebbe essere in grado di immaginarsi lo sviluppo delle macroistruzioni come in un film al rallentatore, in cui poter esaminare ogni dettaglio dei singoli fotogrammi. L'autore ha la capacità di farlo anche nei codici scritti da grandi guru del sistema T_EX.

Paulo Roberto Massa Cereda è l'autore del programma **arara**, la cui versione 3 è correntemente contenuta nelle distribuzioni del sistema T_EX. Al Meeting ci presenta la versione 4, che sarà prossimamente disponibile per tutti tramite le distribuzioni del sistema T_EX. Quindi esclusivamente per noi ci presenterà in video le nuove funzionalità di **arara**. L'articolo ricorda la storia della nascita di questo programma; il video mostra molto chiaramente come si possono usare le sue nuove funzionalità.

Grazia Messineo e Salvatore Vassallo da anni comunicano al G_UIT i loro passi avanti nel creare una raccolta di programmi e di dati per sviluppare test a risposte multiple chiuse e aperte da usare sia come palestre per consentire agli allievi di esercitarsi, sia per sviluppare i test d'ingresso sempre più frequenti in molte università italiane. Con questo particolare intervento gli autori ci presentano come animare certe parti sia testuali sia grafiche dei quesiti proposti come esercizio o come test di

ammissione. Presentano anche come consentire a chi esegue i test di costruire interattivamente curve algebriche che automaticamente si rigenerano se se ne cambiano i parametri.

Il numero e la varietà degli interventi ci dimostrano al di là di ogni dubbio l'interesse che viene dedicato agli aspetti e agli usi meno convenzionali del sistema T_EX; ma ci mostra con evidenza che gli utenti di questo incredibile strumento non si tengono per sé quanto scoprono, ma sono disponibili a farne parte agli altri nello spirito di condivisione delle conoscenze acquisite.

È doveroso e mio piacere personale ringraziare sentitamente l'Università di Trento per averci ospitato; il Dipartimento di Ingegneria informatica e Scienza dell'informazione per averci messo a disposizione l'aula e i mezzi tecnici per questo Meeting. Un grazie particolare a Francesco Endrici che si è prodigato con passione per organizzare questo Meeting.

▷ Claudio Beccari
Professore emerito
Politecnico di Torino
claudio dot beccari at gmail
dot com

I calcoli matematici in pdf_{tex}

Claudio Beccari

Sommario

Il motore di composizione pdf_{tex} così come i motori x_{etex} e lua_{tex} possono fare calcoli ma né il primo, né gli altri compilatori sono stati pensati per farne di diversi da quelli relativamente semplici necessari per eseguire la composizione tipografica. Questo articolo cerca di esporre capacità e limiti delle funzionalità di calcolo ed indica alcuni metodi di calcolo numerico che differiscono non poco da quelli normalmente indicati nei testi di calcolo numerico.

Abstract

The typesetting engine pdf_{tex}, as well as the other typesetting engines x_{etex} and lua_{tex} are capable of some computation functionalities different from those often simple ones necessary to the typesetting requirements. This tutorial is intended to show such computing capabilities and their limits; it also shows some numerical methods that are rather different from those normally described in textbooks on numerical computation.

1 Introduzione

Che un programma per calcolatore sia in grado di fare calcoli non dovrebbe stupire nessuno. Che il programma iniziale creato da Knuth attorno al 1978 facesse qualche calcolo pare del tutto evidente.

La cosa strana per un programma per elaboratore nel 1978, e ancora per diversi anni, era che ogni fabbricante di *mainframe* (perché allora i PC e gli Apple non esistevano ancora) non seguiva nessuno standard o per lo meno seguiva i propri.

Non sto a ricordare le date in cui sono stati introdotti alcuni standard; posso dire che il primo *mainframe* su cui ho messo le mani codificava i suoi dati in parole che oggi potremmo definire di 40 bit e usava una codifica dei numeri in decimale (o piuttosto in biquinario) codificato dove i quattro bit di ogni quaterna avevano i pesi 5, 4, 2, 1. Invero correva l'anno 1964 e nei quattordici anni prima della nascita di T_EX78 erano successe molte cose. Tuttavia ho avuto occasione di lavorare su macchine in cui i byte potevano essere di 7 o di 8 o di 9 bit.

Knuth aveva una giusta preoccupazione: quella che il suo programma avrebbe dovuto girare su quasi tutte le macchine più diffuse; per questo era necessario limitarsi alle operazioni con i numeri interi; anche questi d'altra parte erano codificati in "big-endian" o "little-endian", in parole di 16

oppure 32 bit. Insomma anche nel 1978 le cose non erano stabilizzate.

In questo modo tutta l'aritmetica interna di ogni programma di compilazione del sistema T_EX ricorre ai numeri interi; fu una decisione saggia? Sotto certi aspetti fu una decisione molto saggia; secondo la mia opinione personale oggi non lo è più, tanto che non solo lua_{tex} può fare calcoli in *floating point* (a virgola mobile, come diremmo in italiano) ricorrendo al linguaggio Lua, ma ogni distribuzione relativamente recente del sistema T_EX contiene molti pacchetti in grado di fare calcoli con numeri fratti o *floating point* (pacchetto `fp`, MEHLICH (1999)). Nello sviluppo del futuro L^AT_EX3 esiste già un modulo `l3fp` per svolgere calcoli con numeri *floating point* e con rappresentazione scientifica. Molti pacchetti contengono le proprie librerie di macro per fare calcoli con numeri fratti, ma spesso sono costretti a ricorrere alle operazioni elementari predisposte da Knuth stesso fin dalle prime versioni del suo programma `tex`; questo continua ancora oggi ad essere la pietra di paragone con cui confrontare qualunque altro motore di composizione perché abbia il diritto di contenere la stringa `tex` nel suo nome.

Per conservare valori numerici Knuth aveva essenzialmente preso in considerazione due entità: i numeri e le lunghezze; queste ultime sono poi separate in lunghezze rigide e lunghezze elastiche, ma la differenza era solo che le seconde erano dotate anche di due "apposizioni" costituite dall'ammontare della contrattilità e della allungabilità; agli effetti di questo articolo ci interessano gli interi e le lunghezze rigide.

Questi due tipi di quantità numeriche sono conservabili in due tipi di registri diversi, quelli numerici e quelli dimensionali; i primi sono detti contatori; i secondi non hanno un nome solo, ma nel gergo di T_EX sono chiamati *dimen register*. Entrambi questi tipi di registri sono "parole" di 4 byte di 8 bit, quindi di 32 bit; entrambi possono contenere quantità positive e negative; i registri dimensionali usano un altro bit per distinguere certe lunghezze speciali.

Ne segue che in valore assoluto un contatore può contenere un numero intero non superiore a $2^{31} - 1 = 2\,147\,483\,647$ che sembra sufficientemente grande da poter soddisfare ogni esigenza di calcolo "tipografico".

I registri dimensionali contengono le lunghezze espresse in *scaled point*; uno *scaled point* è la parte $1/2^{16}$ di un punto tipografico e viene conservato

nel registro sempre come numero intero di scaled point. Tenuto conto dei due bit riservati al segno e alle altre informazioni, il massimo numero di scaled point che possono essere inseriti in un registro dimensionale è $2^{30} - 1 = 1\,073\,741\,823$; questo valore corrisponde in realtà solo a circa 16 383.999 98 pt che a sua volta corrisponde a circa 5758.317 mm, vale a dire poco meno di sei metri. Anche questa è una grandezza sufficientemente grande per il calcolo tipografico, ma non è grandissima. Per evitare sorprese di overflow il valore massimo di 16 383.999 98 pt viene conservato nel registro dimensionale identificato con il nome molto esplicito `\maxdimen`; T_EX considera questo valore una specie di “infinito” in un insieme di numeri finito.

I due tipi di registri possono scambiarsi i loro contenuti; nel senso che entrambi contengono solo numeri interi e quelle file di bit possono quasi sempre essere trasferite da un tipo di registro all’altro; il quasi è d’obbligo, perché i più grandi numeri interi hanno 31 bit contro 30 bit delle lunghezze.

Non solo, ma né l’un tipo di registro né l’altro accetta numeri fratti. In realtà i registri dimensionali possono essere visti sia come valori interi di scaled point, sia come valori fratti di punti tipografici con la “virgola”, la quale separa la parte intera dalla parte fratta ed ha alla sua destra 16 posizioni binarie fisse. I registri lunghezza possono essere quindi visti come numeri a virgola fissa. Ma sedici cifre binarie corrispondono a circa 4.8 cifre decimali, quindi trasformando i contenuti dei registri in valori leggibili dalle persone otteniamo numeri decimali con poco meno di 5 cifre decimali corrette.

Fin dall’inizio del sistema T_EX le operazioni eseguibili sui contenuti dei registri numerici o dimensionali potevano essere solo la somma algebrica, la moltiplicazione e la *divisione intera*. Questo va sottolineato: la divisione intera restituisce solo la parte intera del quoziente, non fornisce un quoziente fratto.

Certo una macchina calcolatrice con funzionalità limitate dalla natura interna dei suoi operandi non è granché; anzi direi che si tratta di una calcolatrice molto modesta.

Va detto inoltre che i soli numeri fratti accettati dai motori di composizione sono i fattori moltiplicativi delle lunghezze. Questi sono usati spessissimo senza fare particolare attenzione al fatto che si stia usando una specie di eccezione alla regola; per esempio, sono usati quando durante l’importazione di una figura viene specificato `width=0.7\textwidth` fra le opzioni di `\includegraphics`.

2 Le operazioni di base

Le operazioni di base di tutti i motori di composizione accettano solo `\advance` per eseguire una somma algebrica, `\multiply` per eseguire una mol-

tiplicazione, `\divide` per eseguire una divisione intera.

Gli operandi per `\advance` dovrebbero essere della stessa natura, o entrambi numeri o entrambi lunghezze; il risultato dell’operazione viene messo nello stesso registro del primo operando che viene perciò sovrascritto e il contenuto precedente viene quindi “perso”. Tipico è il modo con cui viene definita la larghezza del margine sinistro:

```
\@tempdima= \paperwidth
\advance\@tempdima by -\textwidth
\oddsidemargin =0.4\@tempdima
\advance\oddsidemargin by -1in
```

Come si vede in questo semplice esempio, singoli comandi come `\advance` non accettano espressioni, ma svolgono una operazione alla volta; su questo punto si tornerà più avanti.

Il codice mostrato sopra usa la sintassi nativa di T_EX, ma con L^AT_EX sarebbe possibile usare i comandi specifici di quel markup come `\setlength` e `\addtolength`; cambierebbe la forma ma non la sostanza.

Un altro esempio è quello con cui si fissa l’altezza della gabbia del testo affinché contenga un numero intero di righe di testo composto con il carattere di default nel corpo normale:

```
\@tempdima=\paperheight
\advance \@tempdima by -2in
\advance\@tempdima by -1.5in
\divide\@tempdima by \baselineskip
\@tempcnta=\@tempdima
\textheight=\@tempcnta\baselineskip
\advance\textheight by \topskip
```

In questo caso vengono sottratti all’altezza della pagina i margini superiore e inferiore e lo spazio (forfettario) da destinare alla testatina e al piedino incluse le distanze di separazione dal testo vero e proprio. Lo spazio rimanente viene diviso per lo scartamento del font normale e il risultato viene poi assegnato ad un contatore. Infine l’altezza del testo viene ottenuta moltiplicando lo scartamento (`\baselineskip`) per il numero intero ottenuto dalla divisione intera eseguita con `\divide`.¹

Questi sono gli usi tipografici delle operazioni fra numeri e lunghezze; pur nella loro modestia sono più che sufficienti per svolgere le funzioni tipografiche richieste dai motori di composizione. Certamente non sono sufficienti per fare calcoli su numeri fratti.

3 L’estensione e-T_EX

Nel 2005 le estensioni introdotte dal *working group* per lo sviluppo del New Typesetting System sono

1. L’esempio è ispirato al codice tratto dal file di opzione per il corpo normale di 10 pt della classe `book`; sono stati sostituiti solo i comandi L^AT_EX `\setlength` e `\addtolength` con i comandi nativi di T_EX.

state introdotte in tutti i motori di composizione, tranne nell'eseguibile knuthiano "puro" $\text{\textit{tex}}$.

Uno dei pochi pacchetti che immediatamente fece uso di queste nuove funzionalità fu proprio la classe $\text{\textit{arstexnica}}$, quella usata per comporre questa rivista; ma non tutte le distribuzioni del sistema $\text{\textit{TeX}}$ furono altrettanto pronte ad incorporare queste nuove funzionalità e si verificarono piccoli inconvenienti di incompatibilità; ma dal 2006 la distribuzione di riferimento di ogni sistema $\text{\textit{TeX}}$ è la $\text{\textit{TeXLive}}$ e da allora non ci sono più stati inconvenienti.

Le nuove funzionalità sono molteplici e si manifestano in diversi ambiti; qui ci interessano le possibilità di calcolo, in particolare i comandi $\text{\textbackslash numexpr}$ e, specialmente, $\text{\textbackslash dimexpr}$ che permettono di svolgere calcoli ricorrendo ad espressioni matematiche numeriche, dimensionali e miste, cosa che con i comandi primitivi del sistema knuthiano non era possibile.

Tanto per fare un esempio, la determinazione dell'altezza delle gabbia interna $\text{\textbackslash textheight}$, con le nuove funzionalità diventa:

```
\textheight=\dimexpr(\paperheight-2in
-1.5in)/\baselineskip*\baselineskip
+\topskip\relax
```

Le nuove funzionalità, oltre alla possibilità di eseguire le quattro operazioni, comportano due novità:

- le divisioni sono sempre divisioni intere, ma il risultato viene arrotondato, non troncato come avviene con i precedenti comandi nativi;
- l'operazione di scalamento avviene conservando il risultato intermedio in un registro di 64 bit, il doppio di un normale registro da 32 bit.

L'operazione di scalamento va meglio spiegata con un esempio. Supponiamo di voler avere una gabbia di stampa con le stesse proporzioni della pagina e supponiamo di conoscere o di avere già specificato la giustezza del testo $\text{\textbackslash textwidth}$. Dobbiamo determinare $\text{\textbackslash textheight}$ in modo che il rapporto altezza su larghezza della gabbia sia uguale al rapporto altezza su larghezza della pagina; dobbiamo cioè determinare

```
\textheight= \paperheight/\paperwidth*\textwidth
```

Semplice e intuitivo; scriviamo infatti la riga:

```
\textheight=\dimexpr
\paperheight*\textwidth/\paperwidth\relax
```

Che cosa succede specificando prima la moltiplicazione e poi la divisione? I due fattori della moltiplicazione sono lunghezze codificate con "parole" di 32 bit le quali contengono i valori delle lunghezze espressi con un numero intero di scaled point; il risultato della moltiplicazione richiede quindi 64

bit per contenere qualunque possibile valore, grande o piccolo, senza doverlo troncato o arrotondare. La successiva divisione per una lunghezza, anch'essa contenuta in una parola di 32 bit, produce un quoziente intero di 32 bit che eventualmente viene arrotondato, invece che troncato; perciò il calcolo dà luogo ad una stringa di bit che rappresentano correttamente una lunghezza in scaled point, ovvero una lunghezza in punti tipografici con 16 cifre binarie fratte.

Cosa distingue questo modo di procedere da quello che si potrebbe ottenere con i comandi primitivi descritti nel paragrafo precedente? Nel secondo caso la prima moltiplicazione produrrebbe un overflow. Infatti una altezza della carta di 297 mm, corrisponde a 845 pt; una larghezza della gabbia, diciamo, di 160 mm corrisponde a 455 pt e il prodotto darebbe luogo ad un valore di 384 475 assai maggiore di $\text{\textbackslash maxdimen}$; quindi l'unica cosa che si otterrebbe è un messaggio di overflow e l'arresto della compilazione.

Cosa succederebbe allora se si eseguisse prima la divisione? Se la larghezza della carta è di 210 mm, che equivale a 597 pt, la divisione intera dell'altezza per la larghezza darebbe il valore $(\text{INT}(845/597) \approx \text{INT}(1.414) = 1$, e si perderebbe ogni informazione utile.

Ecco, questo è un punto chiave; gli scalamenti vanno sempre fatti usando questa nuova funzionalità di $\text{\textit{e-TeX}}$.

4 Alcuni pacchetti che consentono di fare calcoli matematici

Visto che i motori di composizione sono così modesti nel fare i calcoli, già da molto tempo esistono pacchetti che consentono di farli fare al programma di composizione. Tutti questi pacchetti in qualche modo, diretto o indiretto, ricorrono al trucco seguente: ogni numero fratto viene usato come moltiplicatore della lunghezza predefinita $\text{\textbackslash p@}$ che di suo è lunga un punto tipografico; su questa ed altre lunghezze ottenute nello stesso modo si eseguono le operazioni matematiche che si desiderano; il risultato finale viene "mostrato" in forma decimale, dopo avergli tolto le unità di misura. Per esempio, volendo sommare 1,333333 e 2,666666, si potrebbe procedere così:

```
\@tempdima=1.333333\p@
\@tempdimb=2.666666\p@
\edef\@tempA{%
\strip@pt\dimexpr\@tempdima+\@tempdimb}%
\texttt{\@tempA}
```

Il codice di questo esempio produce il risultato seguente: 4. In questo caso il risultato è tondo e il fatto che esso non sia 3.999999 deriva dall'arrotondamento sempre eseguito dall'aritmetica delle espressioni matematiche di $\text{\textit{e-TeX}}$.

Il pacchetto principale noto da molti anni per svolgere questi calcoli è il pacchetto `calc` (THORUP *et al.*, 2014). `calc` non svolge solo le quattro operazioni, ma ridefinisce anche tutti i comandi $\text{\LaTeX}$ che accettano numeri o lunghezze, in modo che siano in grado di analizzare le espressioni matematiche. Siccome molti altri comandi $\text{\LaTeX}$ fanno uso dei comandi ridefiniti, molti altri comandi possono beneficiare delle espressioni che `calc` è in grado di gestire. Si noti inoltre che per usare numeri reali come fattori nelle moltiplicazioni e nelle divisioni bisogna racchiuderli negli argomenti di `\real` oppure `\ratio` in modo da poterli usare convenientemente nelle espressioni matematiche.

Diversi altri pacchetti usano loro macro interne per eseguire i calcoli; per esempio l'estensione `pict2e` (GÄSSLEIN *et al.*, 2014), già documentata da Leslie Lamport nella sua seconda edizione del manuale di $\text{\LaTeX}$ del 1994 ma non effettivamente resa disponibile fino almeno al 2003, ha la sua collezione di macro necessaria per realizzare le varie estensioni annunciate circa 10 anni prima.

I pacchetti `pgf`, `tikz` (TANTAU, 2013) e `pgfplots` (FEUERSÄNGER, 2015) dispongono di loro macro interne e di librerie complete per il calcolo di numerose funzioni matematiche; anzi, `pgfplots` contiene delle raffinate librerie matematiche che consentono di calcolare le funzioni matematiche in modo decimale, con un numero elevato di cifre significative; esse fanno uso del pacchetto `fp`, per eseguire calcoli in floating point. Certamente per usare questi metodi di calcolo bisogna rinunciare alla velocità di esecuzione e si corre anche il rischio che la memoria principale del programma di compilazione risulti insufficiente. Il problema è noto e nei vari forum, compreso quello del `qJr`, ci sono diversi interventi a questo proposito.

Molto prima di questi pacchetti, nati essenzialmente dopo l'avvento di $\text{\LaTeX 2}_{\epsilon}$, il pacchetto `pictex` (WICHURA, 1988) permetteva a Plain $\text{\TeX}$ di cimentarsi con il disegno di diagrammi, anche semi- o bi-logaritmici, essendo esso in grado di calcolare i logaritmi decimali con sufficiente precisione.

Esistono altri pacchetti in grado di fare calcoli. Sebbene sia poco noto e poco usato, mi piace ricordare il pacchetto `curve2e` (BECCARI, 2015) che estende ulteriormente il pacchetto `pict2e` impiegando in lungo e in largo calcoli con i numeri non solo fratti ma anche complessi e quindi usando coppie ordinate di numeri fratti. Per gestire i numeri complessi `pict2e` usa anche la distanza pitagorica (vedi più avanti) e l'anomalia su quattro quadranti; inoltre le coordinate polari (per altro comunemente usate anche da `tikz` e `pgfplots`). Nell'articolo (BECCARI, 2013) ci sono anche altri spunti per eseguire calcoli con `pdftex`.

Posso citare i pacchetti `picture` (OBERDIEK, 2009) e `xpicture` (FUSTER, 2012). Uno estende l'ambiente `picture` di $\text{\LaTeX}$, già esteso da `pict2e`, accet-

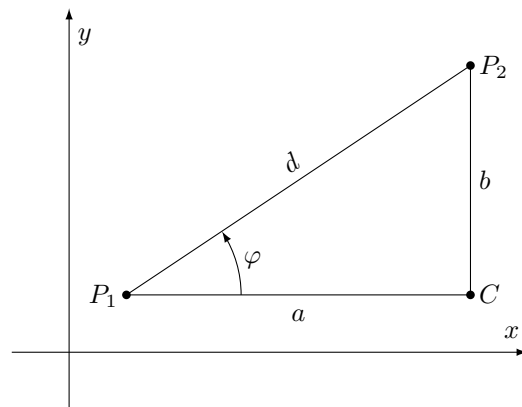


FIGURA 1: La distanza fra due punti

tando coordinate espresse sia nell'unità di misura `\unitlength` tipico dell'ambiente `picture`, sia le unità assolute riconosciute da ogni motore di composizione. L'altro estende le funzionalità disegnando in modo semplice curve normalmente disegnate nei corsi di geometria analitica nelle scuole secondarie superiori, compresi i diagrammi di polinomi e le curve parametriche; esso ricorre alle spline quadratiche di Bézier, che a loro volta richiedono calcoli con numeri fratti.

5 La distanza pitagorica e la radice quadrata

Se non è già definita da macro interne, la distanza fra due punti qualsiasi nel piano (x, y) richiede l'uso del teorema di Pitagora per calcolare la lunghezza dell'ipotenusa di un triangolo rettangolo, figura 1.

Date le coordinate di P_1 e P_2 , non è un problema determinare le lunghezze a e b dei due cateti del triangolo rettangolo della figura 1; si tratta solo delle differenze con segno fra le ascisse o, rispettivamente, le ordinate dei due punti. Sono due numeri fratti con segno, segno che però ai fini del calcolo della distanza d dei due punti è irrilevante. Sia $|a| > |b|$, come rappresentato nella figura; ma senza formalizzare sui simboli, sia $|a|$ la lunghezza del maggiore dei due cateti. È noto allora che la distanza d è data da

$$d = \sqrt{a^2 + b^2} \quad (1)$$

Peccato che non esiste nessun comando nativo di $\text{\TeX}$ che permetta di calcolare quella radice quadrata direttamente e che sia necessario ricorrere ad una macro che permetta di determinarla. Il noto metodo di Newton procede iterativamente; si sceglie un numero di partenza "arbitrario" e si divide il radicando per questo numero; quindi si determina un nuovo numero prendendo la media fra il rapporto appena trovato e il numero usato come divisore. È evidente che una volta trovato il numero giusto rappresentante la radice quadrata, il rapporto e il numero sono uguali e il problema è risolto.

Numericamente parlando dobbiamo ricordare che abbiamo a che fare con una macchina calcolatrice piuttosto scadente e che dobbiamo scegliere un numero conveniente per iniziare il calcolo iterativo; dobbiamo inoltre trovare un test adeguato per decidere quando il risultato è stato trovato.

Per fare questo è necessario riscrivere l'equazione 1 in modo conveniente restringendo il dominio del radicando. Supponiamo, come detto sopra, che sia $|a| > |b|$:

$$d = |a| \sqrt{1 + x^2} \quad (2a)$$

dove

$$x = \frac{b}{a} \quad (2b)$$

In questo modo siamo sicuri che $x < 1$ e che il radicando sia minore di 2; tuttavia se a o b fosse nullo dovremmo procedere diversamente. Il codice diventa pertanto il seguente:

```
\def\distanza#1,#2to#3{%
\def\t@X{#1}\def\t@Y{#2}%
\@tempdima=\t@X\p@
\ifdim\@tempdima<\z@
\@tempdima=-\@tempdima\fi
\@tempdimb=\t@Y\p@
\ifdim\@tempdimb<\z@
\@tempdimb=-\@tempdimb\fi
\ifdim\@tempdima=\z@
\ifdim\@tempdimb=\z@
\def\@X{0}\@tempdimc=\z@
\else
\def\@X{0}\@tempdimc=\@tempdimb
\fi
\else
\ifdim\@tempdima>\@tempdimb
\Divide\@tempdimb by\@tempdima to\@X
\@tempdimc=\@tempdima
\else
\Divide\@tempdima by\@tempdimb to\@X
\@tempdimc=\@tempdimb
\fi
\fi
%-----
\unless\ifdim\@tempdimc=\z@
\unless\ifdim\@X\p@=\z@
\@tempdima=\@X\p@
\@tempdima=\@X\@tempdima
\advance\@tempdima\p@%
\@tempdimb=\p@%
\@tempcnta=5\relax
\@whilenum\@tempcnta>\z@\do{%
\Divide\@tempdima by\@tempdimb to\@X
\advance\@tempdimb \@X\p@
\@tempdimb=.5\@tempdimb
\advance\@tempcnta\m@ne}%
\@tempdimc=\@X\@tempdimc
\fi
```

```
\fi
\edef#3{\strip@pt\@tempdimc}%
\ignorespaces}%
```

La macro `\distanza` è ad argomenti delimitati; i primi due, separati da una virgola, sono due numeri fratti cioè le misure dei due cateti nelle unità dell'ambiente `picture`; il terzo argomento è una control sequence a cui verrà assegnato il risultato e, in quanto rappresentato da un unico token, non necessita di essere racchiuso fra graffe.

Prima della linea di commento vengono eliminati i casi ovvi nei quali uno o l'altro cateto è di lunghezza nulla; viene poi deciso quale sia il cateto maggiore e viene costruito il rapporto chiamato con la macro `\@X`.

Nella seconda parte della macro viene costruito nel registro dimensionale `\@tempdima` il radicando; nel registro `\@tempdimb` viene posta la lunghezza unitaria da usare come valore iniziale delle iterazioni; successivamente viene imposto il contatore `\@tempcnta` con il numero totale delle iterazioni da eseguire sotto controllo del comando del nucleo di L^AT_EX `\@whilenum`. L'unica macro non ancora descritta è `\Divide` ad argomenti delimitati; non è così misteriosa, ma semplicemente si appoggia ad una espressione dimensionale e definisce il terzo argomento (una control sequence) con il valore numerico del quoziente. Quindi

```
\Divide\@tempdima by\@tempdimb to\@X
```

significa “dividi il radicando per l'approssimazione corrente della radice e assegnalo alla macro `\@X`” (il nome è lo stesso, ma la precedente definizione di rapporto dei cateti a questo punto non serve più).

Si noti che questo semplice calcolo diventa molto utile se si vogliono ruotare oggetti senza ricorrere a pacchetti esterni. Infatti i rapporti a/d e b/d fra i cateti e l'ipotenusa del triangolo rettangolo della figura 1 non sono altro che il coseno e il seno dell'angolo φ . Non è quindi difficile scrivere direttamente la matrice di trasformazione in linguaggio PDF che richiede appunto i seni e i coseni con i segni giusti per eseguire la trasformazione affine costituita dalla rotazione; di nuovo si tratta di eseguire due divisioni fra numeri fratti trasformati in lunghezze e sistemare i due rapporti nelle posizioni giuste della matrice di trasformazione. Visto che questo discorso può apparire fumoso senza un esempio, ecco almeno la sintassi²:

2. Qui si usa la sintassi del linguaggio PDF; per usare il linguaggio PostScript le cose sono molto simili, ma le parole chiave, ridotte a una o due lettere con il linguaggio PDF, sono parole complete nel linguaggio PostScript. Ecco perché nel titolo dell'articolo è stato specificato che si sarebbe discussa la possibilità di eseguire calcoli con il motore di composizione `pdftex`. Invece con `luatex` non è necessario fare nulla di speciale perché il linguaggio Lua può svolgere tutti i calcoli necessari; con `xetex` chi produce il file pdf è una versione modificata del programma `dvipdfm` che richiederebbe un'altra variante ancora.

```
% Salvare la matrice corrente
\special{pdf: q}
% Impostare la matrice affine
\special{pdf: <cos  $\varphi$ > <sin  $\varphi$ > <-sin  $\varphi$ > <cos  $\varphi$ >
<0> <0> cm}
<Oggetto da ruotare>
% Ripristinare la matrice precedente
\special{pdf: Q}
```

Ovviamente si può definire una macro che faccia tutto il necessario senza dover consultare questa sintassi ogni volta. Ma qui questa possibile applicazione è segnalata solo per mostrare come spesso si possono usare calcoli interni svolti dallo stesso motore di composizione.

6 Come usare i registri senza allocarli

Nelle macro esposte sono stati usati registri temporanei già allocati, tutti quelli i cui nomi contengono la stringa `temp`. Nel fare altri conti serve poter usare tanti registri senza doversi preoccupare se siano già stati definiti. Un tipico esempio è quello della definizione di `\Divide`:

```
\def\Divide#1by#2to#3{\bgroup
  \dimendef\Num2254\relax
  \dimendef\Den2252\relax
  \dimendef\@DimA 2250
  \Num=\p@ \Den=#2\relax
  \ifdim\Den=\z@
    \edef\x{\noexpand\endgroup\noexpand
      \def\noexpand#3{\strip@pt\maxdimen}}%
  \else
    \@DimA=#1\relax
    \edef\x{%
      \noexpand\egroup\noexpand
      \def\noexpand#3{\strip@pt
        \dimexpr\@DimA*\Num/\Den\relax}}%
  \fi
  \x\ignorespaces}%
```

In questa macro ad argomenti delimitati, il primo argomento è il dividendo costituito da una lunghezza; il secondo è un'altra lunghezza che costituisce il divisore; il terzo è una macro a cui viene assegnato il numero fratto costituito dal quoziente; le prime due lunghezze è bene che siano conservate in registri dimensionali, ma a questo punto (e in molti altri punti...) è meglio non fare riferimento ai registri predefiniti, ma usare registri “qualsiasi” associando loro un nome di comodo. Qui il nome `\Num` è associato al registro dimensionale 2254 e il nome `\Den` al registro dimensionale 2252; i due numeri sono scelti “a caso” fra le migliaia di registri accessibili con le estensioni di e- $\text{T}_{\text{E}}\text{X}$, che permettono di arrivare ad usarne più di 32 000. Questi, d'altra parte, potrebbero essere usati anche da altri pacchetti, quindi è meglio eseguire queste associazioni dentro un gruppo, farvi dentro tutti i calcoli e poi usare uno dei “dirty tricks”

(sporchi trucchi), descritti da Knuth nell'apposita appendice del $\text{T}_{\text{E}}\text{X}$ book, (KNUTH, 1984, app. D), per portare il risultato fuori dal gruppo.

La macro comincia con un `\bgroup` che apre un gruppo con una graffa implicita. Con il comando nativo di $\text{T}_{\text{E}}\text{X}$ `\dimendef` associamo un nome ad un registro dimensionale; carichiamo poi il registro che identifica il numeratore con la lunghezza unitaria e il registro che identifica il denominatore con la lunghezza che costituisce il divisore; siccome bisogna essere sicuri di non dividere per zero, eseguiamo un test sul denominatore; se questo è nullo possiamo decidere, e qui lo si fa, di non impostare un segnale di errore fatale, ma di associare una specie di valore infinito al quoziente; questo viene fatto mediante la definizione di `\x` con `\maxdimen` (sulla quale torniamo fra poco).

Altrimenti si assegna il dividendo al registro dimensionale `\@DimA` e si calcola il quoziente definendo poi la macro `\x` con il suo valore fratto privato dell'unità di misura. Si noti che l'esecuzione della definizione “espansa” mediante `\edef`, esegue a sua volta tutte le macro contenute nel suo testo sostitutivo, tranne quelle precedute da `\noexpand`; quindi del suo testo sostitutivo viene calcolata solo l'espressione dimensionale sotto forma di scalamento e poi vengono tolti i pt dal risultato. Eseguita questa definizione espansa, si esegue la macro `\x` che per prima cosa chiude il gruppo con `\egroup`, cancellando la sua stessa definizione dalla memoria e ripristinando tutti i valori precedenti contenuti nei registri usati dentro il gruppo; tuttavia anche se la macro `\x` non esiste più quello che si sta eseguendo è il suo testo sostitutivo e quindi il valore del quoziente viene assegnato alla macro che costituisce il terzo argomento di `\Divide`.

Questo modo di operare è particolarmente indicato per eseguire i calcoli. Anche se non sempre è necessario ricorrere alla macro `\x` che cancella se stessa, altrettanto spesso lo è; si evitano così conflitti con altre macro usate nello stesso documento e con le macro usate in altri pacchetti.

7 Trasformare i punti in millimetri

Ecco un piccolo esempio che a me pare abbastanza utile. Si tratta di una piccola macro che trasforma le lunghezze espresse in punti in lunghezze espresse in millimetri. Nonostante il rapporto fra millimetro e punto tipografico sia quasi uguale a 3 (per la precisione a 2,84528), passare dai millimetri ai punti moltiplicando per 3 e viceversa dai punti ai millimetri dividendo per 3, almeno per avere una idea delle grandezze e sia pure con un errore di circa il 5%, non è una cosa impossibile a mente: tuttavia qualche volta occorre un valore più preciso. Perché non farlo calcolare al motore di composizione?

Allora ricordiamo che il punto tipografico usato dal sistema $\text{T}_{\text{E}}\text{X}$ è quello anglosassone che si rapporta al pollice mediante la definizione per la quale

un pollice vale esattamente 72,27 pt; d'altra parte un pollice vale 25,4 mm, quindi queste due costanti vanno usate in modo da semplificare i pollici. Ecco dunque la macro per scrivere una lunghezza in millimetri:

```
\newcommand\pttomm[1]{\bgroup
\dimendef\mmttoinch 3333
\dimendef\pttoinch 3334
\dimendef\inlength 3335
\mmttoinch=25.4\p@ \pttoinch=72.27\p@
\inlength=#1\relax
\edef\x{\noexpand\egroup\strip@pt
\dimexpr\inlength*\mmttoinch/\pttoinch
{noexpand\,}mm}\x}
```

Con questi comandi la larghezza di pagina di questo testo vale 209.99994 mm. Come si vede gli arrotondamenti non proteggono da una diminuzione di precisione dei comandi usati, ma sicuramente nessuno si lamenterebbe di leggere 209,99994 mm, che si capisce al volo, piuttosto che 598 pt che ci lascia indifferenti, visto che non abbiamo una consuetudine giornaliera con i punti tipografici.

8 Le funzioni trigonometriche

Le funzioni trigonometriche sono già codificate nel pacchetto trig (CARLISLE, 1999), caricato di default dal pacchetto graphicx (CARLISLE, 2014). Non sono male ma ritengo che sia meglio rifarsi alle formule parametriche della trigonometria:

$$\sin \theta = \frac{2}{\cot x + \tan x} \quad (3a)$$

$$\cos \theta = \frac{\cot x - \tan x}{\cot x + \tan x} \quad (3b)$$

$$\tan \theta = \frac{2}{\cot x - \tan x} \quad (3c)$$

dove θ è l'angolo in gradi e $x = \theta/114.591559$ è metà dello stesso angolo in radianti.

Perché i gradi? Semplice: i gradi contenuti in un angolo giro o piatto o retto sono un numero intero ed è quindi molto più semplice ridurre l'angolo di cui calcolare la funzione trigonometrica ad un intervallo principale. Inoltre la tangente ha uno sviluppo in frazione continua:

$$\tan x = \frac{1}{\frac{1}{x} - \frac{3}{\frac{1}{x} - \frac{5}{\frac{1}{x} - \frac{7}{\frac{1}{x} - \frac{9}{\frac{1}{x} - \frac{11}{x} - \dots}}}}}} \quad (4)$$

molto semplice da calcolare con una macchina calcolatrice modesta come quella di cui disponiamo e anche molto precisa se si limita il dominio principale all'intervallo $[-45^\circ, +45^\circ]$. Ecco quindi che le

formule parametriche sono particolarmente indicate, anche se la forma espressa nelle formule 3 non è quella familiare che si trova in ogni libro delle scuole secondarie superiori.

Se il modulo dell'angolo in radianti non supera $\pi/4$, l'espressione 4 con una "calcolatrice" migliore potrebbe dare dieci o undici cifre esatte; noi dobbiamo accontentarci al massimo di cinque cifre dopo la virgola e siamo fortunati se ne otteniamo quattro.

Ora non si espone come usare la tangente del mezzo angolo per calcolare il seno, il coseno la tangente dell'angolo intero; si tratta di macro che si scrivono quasi immediatamente partendo dalle equazioni 3. Ma è opportuno che queste macro si occupino: di ridurre l'angolo intero all'intervallo fondamentale $[-\pi/2, +\pi/2]$; dei valori speciali, come i multipli interi di $\pi/2$; dei segni eventualmente necessari fino al momento di chiamare la macro della tangente del mezzo angolo la quale deve restituire direttamente sia il valore della tangente, sia quello della cotangente. Ecco quindi il codice di questa macro per il calcolo della tangente del mezzo angolo.

```
\def\g@tTanCotanFrom#1to#2and#3{%
\Divide 114.591559\p@ by#1to\X@
\@tdB=\X@\p@
\countdef\I=2546 \I=11 \def\Tan{0}%
\@whilenum\I>\z@\do{%
\@tdC=\Tan\p@ \@tdD=\I\@tdB
\advance\@tdD-\@tdC
\Divide\p@ by\@tdD to\Tan
\advance\I-2\relax}%
\def#2{\Tan}\Divide\p@ by\Tan\p@ to\Cot
\def#3{\Cot}\ignorespaces}%

```

Come si vede si tratta di una macro piuttosto breve e questo è dovuto alla possibilità di usare la frazione continua indicata dall'equazione 4. L'iterazione è controllata dal comando \@whilenum, facente parte del nucleo di L^AT_EX, ed inizia dall'assegnazione del valore 11 al contatore \I. Forse è una esagerazione, ma non guasta; di sicuro il tempo perso ad eseguire una o due iterazioni di troppo è trascurabile visto che tutti i calcoli sono sviluppati nell'unità centrale di elaborazione del calcolatore. Per la lettura del codice si tenga presente che \X@ rappresenta l'inverso del mezzo angolo trasformato in radianti; le variabili \@td... devono essere preventivamente allocate per registri lunghezza, oppure associati precedentemente con registri lunghezza di numero "qualsiasi", purché le macro per il calcolo del seno, coseno e tangente provvedano correttamente a confinare queste associazioni dentro un gruppo. Questi sono trucchi già descritti in precedenza.

Non è da sottostimare la semplicità dell'uso della frazione continua; quando è possibile disporre di una frazione continua per approssimare una funzione trascendente, è conveniente usare la frazione

continua piuttosto che polinomi opportuni (generalmente polinomi di Chebyshev) spesso usati per la determinazione delle funzioni trascendenti nel firmware di molti programmi di calcolo.

9 Le spline di Bézier

Le spline di Bézier sono curve parametriche che permettono di spostare un punto $P = (x, y)$ da un punto iniziale $P_0 = (x_0, y_0)$ ad un punto finale $P_1 = (x_1, y_1)$ lungo una traiettoria governata da espressioni polinomiali. Le più comuni sono le spline di primo, di secondo e di terzo grado; dipendono tutte dal parametro t di cui normalmente si considera solo l'intervallo $0 \leq t \leq 1$:

$$\mathcal{B}_1 = P_0(1 - t) + P_1 t \quad (5a)$$

$$\mathcal{B}_2 = P_0(1 - t)^2 + C_2 t(1 - t) + P_1 t^2 \quad (5b)$$

$$\mathcal{B}_3 = P_0(1 - t)^3 + C_a 3t(1 - t)^2 + C_b 3t^2(1 - t) + P_1 t^3 \quad (5c)$$

La spline di primo grado $\mathcal{B}_1$ è semplicemente un segmento che unisce i punti P_0 e P_1 ; chiaramente in questa come nelle altre spline, per $t = 0$ la funzione restituisce le coordinate di P_0 e per $t = 1$ essa restituisce le coordinate di P_1 . Generalmente questa spline non è programmata come linea parametrica perché esistono metodi più diretti per tracciare segmenti di retta.

La spline di secondo grado $\mathcal{B}_2$ è un arco di parabola che unisce P_0 e P_1 con le tangenti nei due punti estremi che sono parallele ai segmenti $\overline{P_0 C}$ e $\overline{C P_1}$; il punto C controlla quindi le direzioni delle tangenti di partenza e di arrivo; l'intera curva giace all'interno del *convex hull* delimitato dal triangolo $P_0 C P_1$; per $t = 0,5$ la funzione fornisce un punto nella mezzeria della bisettrice dell'angolo $\widehat{P_0 C P_1}$.

Le spline di Bézier di secondo ordine sono facili da usare perché se si specificano le direzioni, il punto di controllo è univocamente determinato; esse sono normalmente usate per descrivere i contorni dei caratteri TrueType e per alcuni caratteri OpenType.

Siccome gli archi delle spline di Bézier di secondo grado sono archi di parabole, non possono creare curve con flessi, a meno che il punto di unione di due archi consecutivi non venga fatto coincidere con il punto di flesso.

Le spline di Bézier di terzo grado sono le più versatili, nel senso che al variare di t fra zero e uno, il punto restituito dalla spline si muove lungo una cubica che può presentare anche un flesso; in ogni caso tutta la curva si svolge dentro il *convex hull* formato dal poligono con i vertici nei punti P_0, C_a, C_b, P_1 . Le tangenti nei punti estremi sono parallele ai segmenti $\overline{P_0 C_a}$ e $\overline{C_b P_1}$. Diversamente dalle curve di secondo grado, le spline di terzo grado non sono univocamente determinate dalle tangenti, perché i

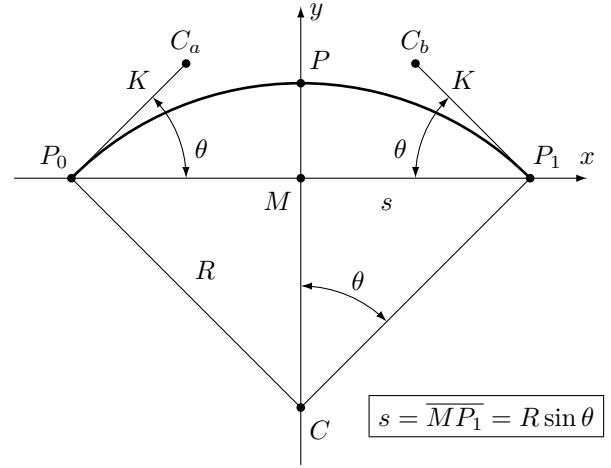


FIGURA 2: Estremi e punti di controllo di una curva di Bézier di terzo grado che approssima un arco di cerchio

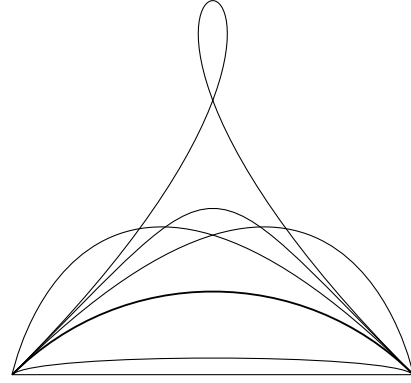


FIGURA 3: Diversi archi di spline di Bézier con gli stessi estremi ma con i punti di controllo via via più distanti dai punti estremi fino a formare un anello

punti di controllo sono due e non uno solo. C'è quindi un margine di arbitrarietà che consente anche di controllare la curvatura dell'arco come è osservabile nella figura 2 dove, scegliendo opportunamente la posizione e la distanza dei punti di controllo dai punti estremi, si riesce a generare un arco indistinguibile ad occhio nudo da un arco di cerchio.

Nella figura 3, invece, si sono scelti per i diversi archi gli stessi punti estremi e le stesse direzioni delle tangenti nei punti estremi, direzioni uguali a quelle dell'arco di cerchio disegnato più scuro; le distanze dei punti di controllo dai punti estremi sono state scelte uguali o diverse; nulle, medie o grandi; gli archi più bassi sono più tesi, fino a confondersi con la loro corda; gli archi superiori asimmetrici hanno o C_a o C_b coincidenti con l'estremo a cui sono associati, ma con l'altro punto di controllo non coincidente col relativo estremo; gli archi superiori simmetrici hanno le distanze dei punti di controllo crescenti fin quando la curva forma un anello.

Questo quindi sottolinea la grande versatilità delle curve di terzo grado, ma mette anche in risalto la difficoltà di scegliere opportunamente i punti di controllo.

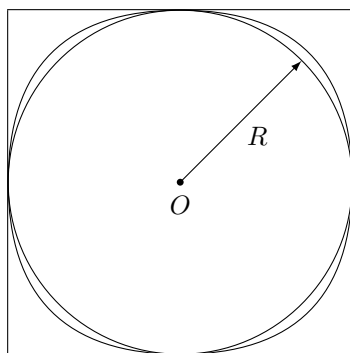


FIGURA 4: due “cerchi” disegnati uno con spline cubiche e uno con spline quadratiche

Fatta questa descrizione delle spline di Bézier, illustro una serie di calcoli che servono per fare una delle altre magie delle spline di terzo grado: disegnare una curva di secondo grado con una spline di terzo grado; questi calcoli devono essere svolti dal motore di composizione; so per certo che il pacchetto `pict2e` li svolge e che la sua estensione `curve2e` sfrutta le stesse macro. Non ho verificato se queste funzionalità e questi calcoli vengano svolti da altri pacchetti, in particolare da `tikz`, `pgf`, `pgfplots` o `xpicture` o da altri pacchetti che non ho mai usato, ma ai fini di questo articolo è irrilevante.

Il problema consiste nel determinare i punti di controllo C_a e C_b di una spline di terzo grado che collega i punti P_0 e P_1 , in modo che la curva diventi esattamente una funzione di secondo grado del parametro t .

Bisogna prendere l’equazione 5c e imporre che i coefficienti di terzo grado si annullino e che quel che resta possa essere messo nella forma dell’equazione 5b. Si tratta di un semplice esercizio d’algebra elementare, ma non dovrebbe sorprendere che una curva di terzo grado possa avere i coefficienti scelti in modo tale da ridursi ad una curva di secondo grado; la soluzione è la seguente:

$$C_a = C + (P_0 - C)/3 \text{ e } C_b = C + (P_1 - C)/3 \quad (6)$$

In altre parole le ascisse e le ordinate dei punti C_a e C_b sono delle semplicissime combinazioni lineari delle corrispondenti coordinate dei punti estremi e di quelle dell’unico punto di controllo della spline di secondo grado; ecco: queste semplici combinazioni lineari si possono fare agevolmente con l’aritmetica descritta in questo articolo e sono alla portata di tutti, purché conoscano come trasformare un numero fratto in una lunghezza e, dopo averci lavorato sopra, sappiano come togliere la stringa `pt` dalla loro rappresentazione decimale.

Curiosità: benché le spline di Bézier di secondo grado siano usate in modo esclusivo per descrivere i contorni dei font TrueType, queste spline non sono così versatili come le spline di terzo grado; quindi

per descrivere un dato contorno è necessario usare più archi quadratici di quanti ne siano necessari con gli archi cubici. Nella figura 4 sono mostrati due “cerchi”, entrambi descritti mediante quattro archi di 90° ciascuno; un cerchio è descritto con le spline di terzo grado e, benché sia solo un’approssimazione di un vero cerchio, è indistinguibile ad occhio nudo da un cerchio esatto; l’altro è descritto con spline di secondo grado e il massimo scostamento dall’altro cerchio ammonta solo a circa il 6% del raggio; ma messi a confronto la differenza è notevole.

Nota Nella lista dei riferimenti bibliografici tutti i manuali indicati fanno parte di ogni distribuzione completa del sistema $\TeX$.

Riferimenti bibliografici

- BECCARI, C. (2013). «Le filigrane e le figure di sfondo». *Ar $\TeX$ nica*, (15), pp. 46–58. URL <http://www.guitex.org/home/numero-15>.
- (2015). *The extension package curve2e*.
- CARLISLE, D. (1999). *The trig package*. $\LaTeX$ Project Team.
- (2014). *Packages in the ‘graphics’ bundle*. $\LaTeX$ Project Team.
- FEUERSÄNGER, C. (2015). *Manual for package PGFPLOTS – 2D/3D plots in $\LaTeX$* .
- FUSTER, R. (2012). *The xpicture package – Several extensions of the picture standard environment*.
- GÄSSLEIN, H., NIEPRASCHK, R. e TKADLEC, J. (2014). *The pict2e package*.
- KNUTH, D. E. (1984). *The $\TeX$ book*. Addison Wesley Publ. Co., Reading, MA.
- MEHLICH, M. (1999). *The fp-package*. Il documento è un file di testo del manuale (UNIX-style).
- OBERDIEK, H. (2009). *The picture package*.
- TANTAU, T. (2013). *TikZ & PGF*.
- THORUP, K.-K., JENSEN, F. e ROWLEY, C. (2014). *The calc package – Infix notation arithmetic in $\LaTeX$* . $\LaTeX$ Project Team.
- WICHURA, M. (1988). *PicTeX*. Il programma è libero, ma il manuale è a pagamento; un sommario abbastanza utile è il documento di Angus Duggan, *PiCTeX command summary*, 1990, disponibile in ogni distribuzione completa del sistema $\TeX$.

▷ Claudio Beccari
claudio.beccari at gmail.com

Il pacchetto ECG

Ezio Aimé, Marco Scavino

Sommario

Nato dalla necessità di avere a disposizione uno strumento in grado di descrivere graficamente qualsiasi condizione elettrocardiografica, il pacchetto ECG qui presentato si propone come ausilio per la creazione di tracciati fittizi, in sostituzione di campioni reali non sempre disponibili nell'archivio di un cardiologo, caratterizzati da buona qualità grafica e uniformità di presentazione a fronte di un'origine elettronica e non umana. Il progetto è presentato in forma ancora molto primitiva in attesa di critiche e suggerimenti costruttivi che stimolino gli autori a perseverare nel loro intento.

Abstract

Developed as a tool to describe graphically any electrocardiographic condition, the ECG package here presented proposes itself as a support to create fictitious graphs which replace those real ones not always available in cardiologists' databases. Furthermore these graphs are characterised by a high graphic quality and a presentation uniformity since they are electronically generated instead, not the result of human inspection. The project is still in progress and is waiting for constructive criticism and useful advices which will spur the authors to reach their goal.

1 Introduzione

Benché la prima registrazione di un elettrocardiogramma (da ora abbreviato in ECG) di un cuore umano integro sia stata effettuata nel 1887 da Augusto Waller, utilizzando un elettrometro capillare a mercurio ed ottenendo una traccia rappresentata da due singole deviazioni distorte, la nascita della tecnica viene attribuita al danese Willem Einthoven che nel 1903 riuscì a migliorare le distorsioni date dall'elettrometro a mercurio introducendo l'uso di un galvanometro da lui stesso progettato. Per ottenere tale registrazione il paziente era invitato ad immergere gli arti in recipienti contenenti una soluzione salina (fig. 1). Da allora i metodi di registrazione si sono raffinati e, soprattutto, standardizzati permettendo di ottenere tracciati confrontabili:

- nel singolo individuo (ricerca di condizioni patologiche intermittenti o evolutive)
- tra vari soggetti (analisi statistica per identificare quadri di normalità o definire morfologie patologiche)

Il "tracciato ECG standard" è ottenuto dalla registrazione di 12 derivazioni di cui 6 agli arti (periferiche) e 6 toraciche (precordiali) ed è articolato in una successione di onde (P, PR, Q, R, S, ST, T) che rappresentano le varie fasi di attivazione elettrica del muscolo cardiaco. Nell'attività di medico-cardiologo si rende necessaria la conoscenza e la capacità interpretativa di tracciati ECG che rappresentano la prima forma di indagine alla quale sottoporre il paziente che si presenta in visita. Possiamo considerare tale esame come una sorta di prima fotografia del nostro paziente e solo con l'esperienza data dalla lettura di innumerevoli tracciati possiamo riconoscere, più o meno rapidamente, la presenza di anomalie che facciano sospettare una possibile situazione di rischio.



FIGURA 1: Metodo utilizzato da Willem Einthoven per registrare i primi ECG

Tale esperienza non va disgiunta dallo studio assiduo della letteratura specializzata in modo da visualizzare e memorizzare quante più possibili variabili (normali o patologiche) che ci permettono di aumentare il nostro bagaglio di conoscenze. In tal senso esistono una moltitudine di testi specializzati e la loro lettura ci consente di giudicare le capacità dei vari autori nel collezionare o reperire esempi esplicativi molto particolari. Talvolta accade che lo stesso tracciato ECG, per caratteristiche assolutamente peculiari, possa comparire in testi di autori diversi; altre volte vengono riprodotte immagini recuperate dal fondo di chissà quale cassetto ed aggiustate alla meglio con programmi di fotoritocco;

innumerevoli volte le tracce sono distorte dalla presenza di artefatti di registrazione. Quello che più spesso si evidenzia è la mancanza di omogeneità nella presentazione delle immagini; notiamo quindi alternanza tra ECG in bianco e nero con ECG colorati, registrazioni su carta millimetrata rossa o verde o azzurra, presentazione di tracce a gruppi di 3 o di 6 o in singole strisciate. Proprio durante la stesura di uno di tali testi mi sono scontrato personalmente con la difficoltà di poter reperire alcuni esempi reali di particolare interesse; nell'occasione ho creato alcuni tracciati "ad hoc" servendomi del pacchetto TikZ. Essendomi ora impegnato nella produzione di una seconda edizione del testo ho pensato, con l'aiuto dell'amico e coautore Marco, di creare un file `archivio.tex` di forme predefinite (una sorta di alfabeto) da unire a piacere per poter creare qualsiasi tipo di nuovo tracciato (dalle lettere dell'alfabeto alle parole). È nato quindi questo pacchetto le cui funzionalità verranno mostrate di seguito.

Nel manuale che sarà allegato al pacchetto, e in parte anche nel presente articolo, abbiamo pensato di articolare il lavoro in una serie di tabelle nelle quali sono rappresentate tutte le varie morfologie d'onda raccolte nel file `archivio.tex` (P, intervalli PQ, Q, R, S, intervallo ST, T ed isoelettrica) che potranno essere utilizzate nella composizione di un tracciato ECG assolutamente personalizzato. Stiamo inoltre ampliando lo stesso `archivio.tex` con l'aggiunta di una serie di morfologie elettrocardiografiche complete (complesso P-QRS-T), tratte da situazioni reali, che potranno essere visualizzate nella loro forma tabulare ed utilizzate in scrittura a seconda dei gusti.

L'elenco che segue ci dà una prima indicazione su come è stata effettuata la codifica dei vari componenti di un complesso ECGrafico

1. p = onda P (di durata tra 60 e 120 ms)

- pp = P positiva
- pn = P negativa
- pd = P difasica
- pb = P bifasica

2. pr = intervallo PR espresso in ms

3. q = onda Q

- q21 - q23 = 20 ms da 1 a 3 mm
- q41 - q45 = 40 ms da 1 a 5 mm

4. r = onda R

- r41 - r45 = 40 ms da 1 a 5 mm
- r62 - r610 = 60 ms da 2 a 10 mm (step di 2 mm)
- r612 - r632 = 60 ms da 12 a 32 mm (step di 4 mm)

- r810 - r845 = 80 ms da 10 a 45 mm (step di 5 mm)

5. s = onda S

- s41 - s45 = 40 ms da 1 a 5 mm
- s62 - s610 = 60 ms da 2 a 10 mm (step di 2 mm)
- s612 - s632 = 60 ms da 12 a 32 mm (step di 4 mm)
- s810 - s845 = 80 ms da 10 a 45 mm (step di 5 mm)

6. st = tratto ST espresso in ms

- st+1 - st+10 = sopraslivellamento da 1 a 10 mm
- st-1 - st-10 = sottoslivellamento da 1 a 10 mm

7. t = onda T

- tp2 - tp10 = positiva da 2 a 10 mm
- tn2 - tp10 = negativa da 2 a 10 mm
- td2 - tp10 = difasica da 2 a 10 mm

8. isoelettrica espressa in ms

La codifica di complessi P-QRS-T completi è generata dall'elenco seguente (assolutamente provvisorio e aggiornabile):

1. prima lettera indicante la tipologia di tracciato ECG

- n = Normale
- e = Extrasistole
- b = Blocco di branca

2. seconda lettera di caratterizzazione del complesso

- n = Normale
- b = Bradicardico
- t = Tachicardico
- v = Ventricolare
- a = Aberrante

3. terza e quarta lettera indicanti la derivazione ECGrafica

- d1 - d3 = D1 - D3
- vr = aVR
- vl = aVL
- vf = aVF
- v1 - v6 = V1 - V6

TABELLA 1: Le 12 derivazioni ECGrafiche possono essere etichettate richiamando il relativo **codice** all'interno della definizione di ECG.

Derivazione	Cod. a	Cod. b	Grafica
D1	!{d1}	?d1	I
D2	!{d2}	?d2	II
D3	!{d3}	?d3	III
aVR	!{vr}	?vr	aVR
aVL	!{v1}	?v1	aVL
aVF	!{vf}	?vf	aVF
V1	!{v1}	?v1	V1
V2	!{v2}	?v2	V2
V3	!{v3}	?v3	V3
V4	!{v4}	?v4	V4
V5	!{v5}	?v5	V5
V6	!{v6}	?v6	V6

2 Onde ECG

Dopo aver dato uno sguardo d'insieme al materiale messo a disposizione del pacchetto ECG entriamo nel dettaglio del suo utilizzo attraverso una più approfondita definizione delle varie strutture presenti; seguirà quindi una breve serie di esempi pratici per terminare con la descrizione tecnica del codice e degli algoritmi che ne permettono il funzionamento.

2.1 Codifica delle derivazioni

Volendo iniziare a descrivere un tracciato ECG, per prima cosa potremo inserire l'etichetta della derivazione a cui si fa riferimento presa dalla tab. 1 e richiamarla con il comando `!{<nome>}` o tramite `? [x shift] <etichetta>`, dove `<etichetta>` sarà il testo che verrà stampato direttamente sul tracciato.

Con la prima opzione si ha la garanzia di uniformità e di standardizzazione delle etichette delle derivazioni mentre con la seconda viene reso possibile l'inserimento di eventuali sigle o testo non espressamente previsto dal database. Subito dopo sarà opportuno inserire un simbolo di taratura richiamato dalla lettera `T<cm>` dove `<cm>` indica i centimetri di altezza dell'onda di taratura (analogo risultato con il comando `!{tara}`).

2.2 Onda P

Veniamo ora alla descrizione delle varie parti di un complesso ECGrafico partendo con la definizione delle onde P

pp 02 80

Le due lettere iniziali indicano un'onda P positiva (con "pn" si indicherà una P negativa, con "pd"

una P difasica e con "pb" una P bifasica). Le prime due cifre del numero esprimono in mV l'altezza per le onde positive (1 mV–3 mV) o la profondità (per le onde negative) mentre le due/tre successive cifre indicano la durata in ms (rispettivamente di 60 ms, 80 ms e 100 ms).

Vengono riportate una serie di onde P positive, negative, bifide e difasiche in tab. 2, presenti nel database¹. Anche in questo caso le tabelle riportano il codice `!{<nome onda>}`, dove `<nome onda>` è il nome presente nel database dell'onda che interessa utilizzare.

2.3 Intervalli isoelettrici

I tratti di isoelettrica da interporre tra:

- onda P ed onda Q (intervallo PQ o PR)
- onda S ed onda T (intervallo ST)
- onda T ed onda P (tratto isoelettrico).

Tali intervalli possono essere espressi in millisecondi ed essere richiamati tramite la dicitura `i{<misura>}`, dove `<misura>` può essere un qualsiasi valore intero, che indica i millisecondi di durata dell'onda isoelettrica.

2.4 Complesso QRS

Onda Q

L'onda Q, che rappresenta la prima deflessione negativa del complesso QRS, può assumere valori di durata fisiologica (10 ms) o patologica (20 ms). Alcune morfologie relative a tale onda sono riportate nella tab. 3(a), accompagnate dal relativo codice di inserimento `q{<ms>}<mm>`. La prima cifra `<ms>` indica le decine di millisecondi (ad esempio 2 indica 20 ms), ossia la durata dell'onda, mentre le successive `<mm>` sono i millimetri di altezza dell'onda.

Onda R

L'onda R, prima deflessione positiva del complesso QRS, viene definita con intervalli di durata che vanno da 30 a 50 ms. Un esempio di come comparire l'onda è riportato in tab. 3(b). In modo simile all'onda Q, il codice per ottenere l'onda R è `r{<ms>}<mm>`, dove la cifra `<ms>` sono le decine di millisecondi e le successive `<mm>` l'altezza dell'onda in millimetri.

Onda S

L'onda S, seconda deflessione negativa del complesso QRS, alla pari delle onde R e Q, può essere richiamata tramite la dicitura `s{<ms>}<mm>`. Si può osservare una serie di onde S riportate in tab. 3(c).

1. In futuro cercheremo di automatizzare anche questa tipologia di onda, più complessa da rappresentare.

TABELLA 2: Esempi di onde P di durata 100 ms.

(a) Positive (sigla pp, altezza 1 mV–3 mV).		(b) Negative (sigla pn, altezza 1 mV–3 mV).		(c) Bifide (pb) e difasiche (pd).	
Codice	Morfologia	Codice	Morfologia	Codice	Morfologia
<code>!{pp0199}</code>	^	<code>!{pn0199}</code>	~	<code>!{pd1199}</code>	^~
<code>!{pp0299}</code>	^	<code>!{pn0299}</code>	~	<code>!{pb2199}</code>	^~
<code>!{pp0399}</code>	^	<code>!{pn0399}</code>	~	<code>!{pb1299}</code>	^~

TABELLA 3: Alcuni esempi di onde del complesso QRS.

(a) Onda Q		(b) Onda R		(c) Onda S	
Codice	Morfologia	Codice	Morfologia	Codice	Morfologia
<code>q21</code>	v	<code>r31</code>	^	<code>s31</code>	v
<code>q22</code>	v	<code>r32</code>	^	<code>s32</code>	v
<code>q23</code>	v	<code>r33</code>	^	<code>s33</code>	v

2.5 Onda T

L'onda T può essere positiva, negativa o difasica². Le onde sono ottenibili tramite la sigla `t` *<deviazione>* *{<ms>}* *{<mV>}*, dove *<deviazione>* è una lettera tra p, n o d (quest'ultima non ancora disponibile), *<ms>* è un argomento che indica le decine di millisecondi e *<mV>* rappresenta l'altezza in millivolt dell'onda T.

2.6 Complessi P-QRS-T

Questa tipologia di complessi (completi e tratti da ECG reali) viene fornita nel database ed è accessibile tramite la sigla `!{<nome onda>}`. Nella tab. 4 sono riportate alcune onde a titolo d'esempio.

TABELLA 4: Onde dei complessi P-QRS-T.

Codice	Morfologia
<code>!{nnd2}</code>	
<code>!{evvf}</code>	
<code>!{n2v3}</code>	
<code>!{wbv6}</code>	

2. Attualmente le forme difasiche non sono ancora state implementate e le attuali forme devono essere corrette per rendere al meglio l'aspetto asimmetrico delle stesse.

3 Esempi pratici

Costruiamo un ECG

Volendo costruire *ex novo* un tracciato elettrocardiografico dovremo posizionare in sequenza le diverse onde ed intervalli che lo costituiscono³ antepo-
nendo al tutto il richiamo di un codice di etichetta e del codice di taratura ed ottenendo la fig 2.

```

1\begin{figure}
2\begin{ecg}
3\ECG{?v1, T1, !{pp0160}, i20, q21,
4r48, s31, i20, tp44, i400}
5\end{ecg}
6\caption{Un primo complesso P-QRS-T.}
7\end{figure}

```



FIGURA 2: Un primo complesso P-QRS-T.

È possibile replicare il codice tramite l'opportuna chiave `*` che prende due argomenti. Il primo è il numero di repliche del tratto interessato, mentre il secondo è la sequenza di onde del tratto che interessa replicare.

```

1\ECG{?v1, T1, *{3}{!{pp0160}, i20,
2q21, r48, s31, i20, tp44, i400}}

```

Lo sfondo millimetrato si adatta automaticamente alle dimensioni del tracciato (vedere fig. 3).

Vediamo un altro esempio (fig. 4) generato dal seguente codice:

3. Nella versione attuale non è ancora inserita la definizione dell'onda U.

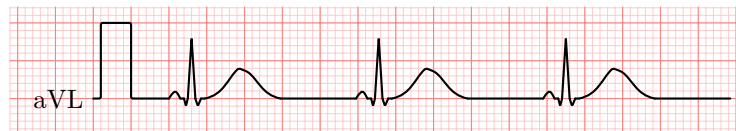


FIGURA 3: Un esempio del medesimo complesso ripetuto tre volte.

```

1 \begin{ecg}
2 \ECG{i40, !{pp0260}, i100, q22, r416,
3 s42, tp48, i200, i20, !{pp0260}, i140,
4 q22, r416, s42, tp48, i160, i60,
5 !{pp0260}, i200, q22, r416, s42, tp48,
6 i100}
7 \end{ecg}

```

dove si nota un progressivo allungamento dell'intervallo PR; il lettore potrà esercitarsi a continuare la costruzione sino ad ottenere un episodio di BAV di II grado tipo Luciani-Wenchebach.

4 Codice

Il pacchetto per creare elettrocardiogrammi è basato principalmente sulle funzionalità fornite dal pacchetto TikZ e del progetto L^AT_EX3. Il primo viene impiegato per disegnare i tracciati, mentre il secondo fornisce gli strumenti per una corretta gestione delle sigle associate ai vari tracciati, in modo che vengano riconosciute e l'onda richiesta sia stampata nelle dimensioni specificate.

4.1 Il comando \ECG

Cuore del pacchetto è il comando `\ECG`. Esso accetta come argomento una lista di onde ECG, disegnate poi nell'ordine in cui sono richiamate. Ogni singola onda può essere disegnata direttamente inserendo l'opportuna sigla che la contraddistingue, oppure attingendo all'`archivio.tex` con la scrittura `!{nome onda}`, che permette di disegnare il complesso presente nel database.

```

1 \ECG[ opzioni TikZ ] { lista di onde }

```

Il comando si appoggia internamente sulla macro `\sk_ECG:nn`, che prende due argomenti. Il primo rappresenta le opzioni di TikZ, che verrà passato poi come opzione del comando `\draw`, mentre il secondo è una lista formata dalle sigle delle onde. Tale lista viene suddivisa rispetto alla virgola grazie a una *sequence* e ogni singolo elemento della lista viene passato al comando `\sk_ECG_use:w` per essere interpretato.

Il comando `\sk_ECG_use:w` prende quattro argomenti.

```

1 \DeclareDocumentCommand \sk_ECG_use:w
2 { 0{ } m u q_nil m }
3 \tl_set:Nn \sk_ECG_onda_tl

```

```

4 { \draw[thick,rounded ~ corners=0.25
5 mm,line ~ cap=round, shift=(
6 sk_end),#4,#1] }
7 \str_case:nn { #2 }
8 {
9 { ! } { \tl_put_right:Nx \
10 sk_ECG_onda_tl { \
11 cs_if_exist_use:cF { sk_ECG_ #3
12 } { ; } } }
13 { ? } { \sk_ECG_onda_label:nw #3 \
14 q_nil }
15 { i } { \sk_ECG_onda_iso:w #3 \
16 q_nil }
17 { p } { \sk_ECG_onda_p:nnnw #3 \
18 q_nil }
19 { q } { \sk_ECG_onda_q:nw #3 \q_nil
20 }
21 { r } { \sk_ECG_onda_r:nw #3 \q_nil
22 }
23 { s } { \sk_ECG_onda_s:nw #3 \q_nil
24 }
25 { t } { \sk_ECG_onda_t:nw #3 \q_nil
26 }
27 { T } { \tl_put_right:Nx \
28 sk_ECG_onda_tl { (0,0) -- (1mm
29 ,0) -- (.1,1) -- (.5,1) --
30 (.5,0) -- (1,0) } }
31 { * } { \sk_ECG_onda_ripeti:nw #3 \
32 q_nil { #4 } }
33 }
34 \tl_put_right:Nn \sk_ECG_onda_tl {
35 coordinate (sk_end) {} ; }
36 \sk_ECG_onda_tl
37 \bool_if:NT \sk_ECG_start_bool { \
38 bool_set_false:N \
39 sk_ECG_start_bool }
40 }

```

Di seguito sono illustrati nel dettaglio i vari argomenti.

1. Il primo argomento e il quarto sono rispettivamente le opzioni per la singola onda o l'intero tracciato. Vengono passati al comando `\draw`, che è salvato dentro alla *token list* `\sk_ECG_onda_tl`. In tal modo è possibile assegnare opzioni all'intero tracciato o al singolo complesso (per esempio colorare un'onda per evidenziarla).
2. Il secondo argomento rappresenta la prima lettera della sigla, che sarà interpretata per ottenere l'onda corretta o per accedere ad alcune funzionalità. Per ottenere ciò, si sfrutta il comando `\str_case:nn` che premette di confrontare tale argomento con una serie di



FIGURA 4: Un esempio di progressivo allungamento dell'intervallo PR tipico del blocco atrio-ventricolare di II grado tipo Möbitz 1.

valori, utilizzando poi il codice nel caso in cui l'argomento coincida.

Attualmente sono disponibili le chiavi

- ? `[ $\langle x shift \rangle$ ] $\langle etichetta \rangle$` , che permette di stampare nel tracciato l' $\langle etichetta \rangle$ il cui testo è l'argomento evocando il comando. Può essere spostata del valore opzionale $\langle x shift \rangle$.
- ! $\langle nome onda \rangle$, con cui si può accedere all'onda definita con $\langle nome onda \rangle$ tramite il comando `\nuovoECG`;
- le lettere `q`, `r`, `s` e `t` permettono di disegnare le rispettive onde (anche la lettera `p` è disponibile, ma deve essere ancora ottimizzata);
- il simbolo `* { $\langle numero ripetizioni \rangle$ } { $\langle lista picchi \rangle$ }` permette di replicare la $\langle lista picchi \rangle$ tante volte quante indicate in $\langle numero ripetizioni \rangle$.
- La lettera `i` $\langle millisecondi \rangle$, per disegnare una onda isoelettrica di durata (in millisecondi) pari al valore $\langle millisecondi \rangle$.
- La lettera `T` $\langle altezza picco \rangle$ permette di inserire un'onda di taratura di altezza $\langle altezza picco \rangle$ centimetri.

- Il terzo argomento è delimitato dal comando `\q_nil` e raccoglie i valori di durata delle onde e la loro altezza passandoli alle varie macro selezionate dal secondo argomento. Ciascuna macro gestisce poi internamente in maniera opportuna i propri parametri: alcune, per esempio, cercano un'ulteriore lettera per distinguere tra casi di onde con morfologie diverse. Un esempio è l'onda `p` che accetta ulteriormente una lettera `n` (onda negativa), `p` (onda positiva) e `d` per onde difasiche e `b` per onde bifide.

4.2 L'ambiente `ecg`

Il comando `\ECG` può essere impiegato dentro a un opportuno ambiente `ecg`, che, oltre a permettere di disegnare più tracciati consecutivi uno sotto l'altro, vi colloca dietro una griglia millimetrata, le cui dimensioni sono automaticamente determinate dalla larghezza dei tracciati.

Questo è reso possibile sfruttando due funzionalità del pacchetto TikZ: il nodo speciale `current bounding box` e il livello `sk@back` definito tramite il comando `\pgfdeclarelayer` e posto dietro al livello principale impostando la lista `\pgfsetlayers{sk@back,main}`.

È quindi possibile andare a disegnare la griglia millimetrata ponendo i comandi all'interno dell'ambiente `pgfonlayer`. Si definiscono quindi le due coordinate `grid_start` e `grid_stop`, che dipendono dall'attuale `bounding box` dei tracciati. Tali coordinate sono spostate rispetto ai valori impostati con le chiavi apposite `grid bottom`.

```

1\begin{pgfonlayer}{sk@back}
2 \coordinate(grid_start) at ([xshift
   =-\sk_ECG_gridlf_tl, yshift=-\
   sk_ECG_gridbt_tl] current ~
   bounding ~ box.south ~ west);
3 \coordinate(grid_stop) at ([xshift=\
   sk_ECG_gridrg_tl, yshift=\
   sk_ECG_gridtp_tl] current ~
   bounding ~ box.north ~ east);
4 \draw[color=red!10,step=0.1cm,very ~
   thin] (grid_start) grid (
   grid_stop);
5 \draw [color=red!25,step=.5cm](
   grid_start)grid(grid_stop);
6\end{pgfonlayer}

```

All'interno di tale ambiente il comando acquisisce un ulteriore argomento facoltativo tra parentesi tonde, che permette all'utente di decidere la posizione del tracciato.

```

1\begin{ecg}
2\ECG [opzioni TikZ] (posizione) {
   lista di onde }
3\end{ecg}

```

Tutti i complessi ECG del database sono definiti nel file `archivio.tex`, caricato automaticamente dal pacchetto. Il database è liberamente ampliabile, utilizzando l'apposito comando `\newECG`, che prende due argomenti: il primo rappresenta la sigla dell'onda che si sta creando e che servirà per poterla richiamare successivamente all'interno del comando `\ECG`, mentre il secondo argomento, rappresentato dalle coordinate TikZ, verrà poi impiegato per disegnare la forma d'onda. Tale disegno deve necessariamente partire dalla coordinata `(0,0)`.

5 Conclusioni

Il pacchetto ECG nasce con l'intenzione di fornire una base di supporto per la creazione di tracciati elettrocardiografici tramite un'interfaccia semplice, che permetta di disegnare facilmente le varie tipologie di onde tipiche di questi tracciati. È un progetto ancora allo stadio embrionale, ma i suoi autori

si impegneranno per svilupparlo, implementando nuove tipologie di onde e nuove funzionalità.

- ▷ Ezio Aimé
ezio dot aime at fastwebnet dot it
- ▷ Marco Scavino
scavino dot marco93 at gmail dot com

Canzonieri online con L^AT_EX

Francesco Endrici, Matteo Conci

Sommario

La creazione di un canzoniere (qui intendiamo con canzoniere una raccolta di canzoni, che possono avere o meno indicazioni musicali, come accordi, annotazioni e indicazioni sulla metrica) è un'operazione tipograficamente complicata, per la complessità delle strutture che esso porta con sé. Vedremo come è possibile usare il pacchetto **songs** per facilitare il lavoro e infine presenteremo un nuovo sistema per lavorare online.

Abstract

The production of a songbook (a book where songs are included with or without musical notation like chords and measures) is typographically a difficult issue because of the complexity of the structures that it contains. We describe how is possible to use the **songs** package to ease the work and then we present a new system for the online work.

1 Introduzione

La creazione di un canzoniere è un lavoro che viene svolto molto più spesso di quanto si possa immaginare. Nei gruppi scout capita spesso che ci sia qualcuno che vuole cimentarsi nella riscrittura del canzoniere di Gruppo, nelle parrocchie i cori creano il proprio canzoniere per aiutare l'assemblea a partecipare al canto. Purtroppo mediamente i lavori eseguiti da persone con limitate competenze tipografiche risultano di cattiva fattura e, cosa ancora più grave, di difficile interpretazione per cantori e suonatori.

Il processo che solitamente viene eseguito da chi compone il canzoniere prevede, dopo aver definito la lista delle canzoni, di cercarne i testi su internet o nel proprio database personale e fare un copia-incolla nel programma di videocomposizione utilizzato. A questo punto, solitamente, iniziano i problemi: bisogna dare una coerenza a tutte le canzoni, assegnare un font comune, fare in modo che tutti i ritornelli abbiano un determinato font, e così gli accordi. Qualcosa può sfuggire anche al più attento compositore.

E poi c'è il grosso problema della *posizione* degli accordi: un accordo dovrebbe trovarsi sopra alla sillaba in cui si inizia a suonarlo: sillaba e accordo dovrebbero essere entità inscindibili. Questo diventa molto difficile con i normali sistemi di videoscrittura, se non impossibile. Infatti succede che ogni volta che si cambia font al documento si ha uno slittamento relativo fra le righe degli

accordi e quelle del testo, vanificando il lavoro di allineamento precedentemente effettuato. Un modo per risolvere il problema è l'uso di font non proporzionali, dove ogni lettera, spazio, simbolo hanno la medesima larghezza, stratagemma utilizzato anche dai più seri siti che pubblicano canzoni con accordi.

2 Il pacchetto **songs**

Esistono diversi pacchetti L^AT_EX per la composizione di canzonieri, come Songbook e **songs**¹. Il pacchetto **songs** di Kevin H. Hamlen è quello che a nostro parere offre la maggior completezza e flessibilità, in particolare per la semplicità nell'inserimento degli accordi, la possibilità di "memorizzare" gli accordi per ripeterli in diverse strofe e, soprattutto, la possibilità di trasporre automaticamente gli accordi.

Una canzone può essere solitamente strutturata secondo uno schema ben definito: contiene strofe, ritornelli, accordi, indicazioni (ad esempio il numero di volte che un ritornello deve essere ripetuto), parti strumentali e un blocco con informazioni come titolo e autore. Pertanto è facile anche pensare a dei comandi e degli ambienti che permettano di assegnare a ognuno di essi la corretta formattazione.

La struttura tipica di una canzone è quella indicata nel codice della figura 1:

- una canzone inizia con `\begin song` e termina con `\end song`;
- `\begin song` ha come argomento obbligatorio il titolo della canzone;
- `\begin song` ha diversi argomenti opzionali, come il nome dell'autore, le informazioni sul copyright ed eventuali riferimenti biblici, per le canzoni religiose;
- un ritornello è delimitato da `\begin chorus` e `\end chorus`;
- una strofa è delimitata da `\begin verse` e `\end verse`. La versione asteriscata evita la numerazione delle strofe (da usare ad esempio con canzoni con una sola strofa);
- gli accordi si inseriscono in una coppia di parentesi quadre preceduta da una barra rovesciata `\[`;

1. <http://songs.sourceforge.net/>

```
\beginsong{Alleluia Passeranno i cieli}
[by={Costa, Varnavà}]
\beginchorus
\{D\}Alle\{A\}luia, \{B-\}alleluia,
\{F#-\}alleluia,
\{G\}alleluia, \{D\}allelu\{E-\}ia, \{A\}
\{D\}alle\{G\}luia, al\{A\}lelu\{D\}ia.
\endchorus
\beginverse*
\{D\}Passeranno i \{A\}cieli
e \{B-\}passerà la \{F#-\}terra,
\{G\}la Tua parola \{D\}non passe\{E-\}rà.
\{A\} \{D\}Alle\{G\}luia, al\{A\}lelu\{D\}ia.
\endverse
\endsong
```

FIGURA 1: Struttura tipica di una canzone scritta con il pacchetto **songs**.

- è possibile utilizzare qualsiasi carattere per inserire un accordo, purché poi si provveda a impostare opportunamente i comandi di **songs**. Noi abbiamo scelto la notazione anglosassone, per la velocità di inserimento (La=A, Si=B, Do=C, ...);
- un bemolle si inserisce con & e un diesis con #.

Una caratteristica un po' più complicata, ma molto pratica, è la possibilità di memorizzare e ripetere gli accordi in diverse strofe, come si vede nel codice della figura 2.

songs memorizza automaticamente gli accordi della prima strofa di una canzone. Per poterli poi inserire in un'altra strofa sarà sufficiente scrivere, al posto giusto, un segno di apice, ^. Se si vogliono ripetere degli accordi di una strofa diversa dalla prima (o se si è usato l'ambiente **verse** per scrivere un'introduzione) si può usare il comando **\memorize** che dice a **songs** di cominciare a memorizzare gli accordi a partire dal punto in cui viene inserito.

songs permette di memorizzare molti set di accordi e di richiamarli, tramite la definizione di contatori personali.

Spesso le canzoni hanno un'introduzione o una chiusura strumentale. **songs**, per scelta dell'autore, non ha degli ambienti specifici per queste parti, pertanto occorre crearseli con ciò che mette a disposizione.

Per le introduzioni si può usare il codice:

```
\ifchorded
\beginverse*
\vspace*{-0.8\versesep}
{\nolyrics \{D\}\{F#-\}\{D\}\{F#-\}}
\vspace*{-\versesep}
\endverse
\fi
```

```
\beginsong{Nella Tua presenza}
[by={Ricci}]
\beginverse
\{D\}Nella tua presenza
avvolti da \{F#-\}Te,
\{D\}nella tua dimora
insieme con \{F#-\}Te,
con la vita Tua che \{G\}sboccia
nell'anima, in \{F#-\}noi,
con la linfa tua, la \{G\}stessa,
in ciascuno di \{B-\}no\{A\}i.
\endverse
\beginverse
^Eccoci fratelli, parte di ^Te,
^eccoci famiglia, una sola con ^Te,
che risorto dai la ^vita
che non muore ^mai
che risorto dentro al ^cuore
accendi il tuo ^cie^lo. \{F\}
\endverse
...
\endsong
```

FIGURA 2: Ripetizione degli accordi in diverse strofe.

\ifchorded dice a **songs** che il codice che segue va inserito solo in un canzoniere con accordi. **\vspace*{-0.8\versesep}** riduce la spaziatura verticale altrimenti troppo abbondante. **{\nolyrics ...}** indica che si sta scrivendo una parte solo strumentale, senza parole. **\fi** chiude **\ifchorded**. Una struttura del tutto simile può essere utilizzata per gli intermezzi e i finali.

Dal punto di vista delle personalizzazioni tipografiche **songs** permette di fare praticamente tutto quel che si vuole. Sono già definiti tutti i comandi necessari a modificare l'aspetto delle strutture più importanti:

\lyricfont imposta il font di tutta la canzone

\versefont imposta il font delle strofe

\chorusfont imposta il font dei ritornelli

\printchord imposta il font degli accordi

Il pacchetto **songs** offre anche la possibilità di creare tre differenti indici:

- Indice dei titoli
- Indice degli autori
- Indice dei riferimenti biblici

Per poter generare il primo indice è ovviamente necessario specificare il titolo delle canzoni all'interno dell'argomento obbligatorio del comando **\beginsong**. Se una canzone fosse conosciuta con diversi titoli, sarebbe possibile inserire tutti i titoli nell'indice usando il comando **ititle={Titolo alternativo}** all'interno dell'argomento opzionale di **\beginsong**.

L'indice degli autori viene creato a partire dagli argomenti dati alla chiave `by={Autori}` all'interno dell'argomento opzionale di `\beginsong`. I nomi di diversi autori devono essere inseriti separati da virgola, punto e virgola oppure da una parola specificata con il comando `\authseppword`.

L'indice dei riferimenti biblici viene creato a partire dagli argomenti dati alla chiave `sr={John 3:16,17, 4:1-5; Jude 3}` all'interno dell'argomento opzionale di `\beginsong`.

`songs` non usa i classici sistemi offerti dalle distribuzioni L $\text{\TeX}$ per creare gli indici, cioè *MakeIndex* o *xindy*, ma usa un programma scritto da Hamlen chiamato `songidx`. Questo programma è scaricabile assieme al pacchetto `songs` dal sito dell'autore e installabile su tutte le piattaforme più note.

3 La classe `canzoniereonline.cls`

Le potenzialità del pacchetto `songs` sono state sfruttate per la creazione di un sistema automatico di creazione di canzonieri, che possa permettere anche ai meno esperti di ottenere un prodotto di alta qualità tipografica. È stata scritta quindi una classe, chiamata `canzoniereonline`², che offre la possibilità di scegliere fra diverse impostazioni per creare il proprio canzoniere su misura.

3.1 Formati

La classe prevede diversi formati di stampa:

standard opzione di default che genera un canzoniere in formato A4 verticale;

a5vert genera un canzoniere in formato A5 verticale;

a5oriz genera un canzoniere in formato A5 orizzontale;

a6vert genera un canzoniere in formato A6 verticale.

Dal punto di vista della scelta dei caratteri si sono definite soltanto due possibilità: l'opzione `palatino` carica il pacchetto `mathpazo` e di conseguenza il font Palatino, mentre l'opzione `libertine` carica il pacchetto omonimo per sfruttare il font Linux Libertine.

Rispetto al pacchetto `songs` la classe `canzoniereonline` offre nuove possibilità. Ad esempio è possibile modificare a proprio piacimento il modo di rappresentare gli accordi "minori". Questi solitamente vengono rappresentati o con una *m* o con il simbolo $\flat$. La classe rende attivo il carattere $\flat$ all'interno degli accordi e lo sostituisce con il comando `\canzononline@myminor`. Va da sé che gli accordi devono necessariamente essere scritti usando il segno $\flat$ per indicare i minori, altrimenti si perde questa possibilità.

2. <https://github.com/franen/canzoniereonline>.

3.2 Indici

La creazione degli indici eseguita con il programma `songidx` è pensata per canzonieri di lingua inglese e non è utilizzabile nel caso di canzoni che inizino con lettere accentate. Per questo motivo, e anche per "liberarsi" dalla necessità di installare un ulteriore programma sul proprio calcolatore, si è lavorato per poter compilare gli indici utilizzando *MakeIndex* o *xindy*.

Anche il primo, si sa, non si comporta bene con le lettere accentate, ma grazie a un trucchetto si riesce a ottenere il risultato corretto. *xindy* invece è stato appositamente sviluppato per lavorare con caratteri particolari, ma non interagisce bene con i file `.idx` generati da pdfL $\text{\TeX}$, mentre dà il meglio di sé con X $\text{\TeX}$.

Se un canzoniere contiene una canzone dal titolo "È la vita" succede che il comando `\index{È la vita}` genera, usando pdfL $\text{\TeX}$, una voce nel file `.idx` strutturata in questo modo: `\indexentry{\IeC {\‘E} la vita}{1}` dove le lettere accentate diventano argomento di `\IeC`. Lanciando poi *MakeIndex* si ottiene un file `.ind` contenente la stringa `\item \IeC {\‘E} la vita, 1`, che però viene indicizzata fra i simboli, prima della lettera A.

Per avviare a questo problema bisogna fare in modo che le lettere accentate vengano riconosciute e ordinate correttamente; si può applicare il codice scritto da Enrico Gregorio³ che permette di trasformare una voce `\index{È}` in una `\index{e@È}`, che preserva il corretto ordine delle parole nell'indice.

```
\makeatletter
\def\accents@list{\do\IeC\do\‘\do\’}
\def\letters@list{\do\i{i}}
\newcommand{\noaccentindex}[1]{%
  \begingroup
  \protected@edef\next{#1}%
  \ifnum\pdfstrcmp{%
    \detokenize\expandafter{\next}}%
    {\detokenize{#1}}=\z@
    \endgroup\index{#1}%
  \else
    \def\do##1{\def##1####1{####1}}%
    \accents@list
    \def\do##1##2{\def##1{##2}}%
    \letters@list
    \edef\next{\endgroup}%
    \def\noexpand\next{\next}\next
    \expandafter\index\expandafter%
    {\next @\detokenize{#1}}%
  \fi}
\makeatother
```

3. <http://www.guitex.org/home/forum/5-tex-e-latex/58220-makeindex-e-lettere-accentate?lang=it#58225>.

L'uso di xindy permette di ovviare a questo problema dal momento che i caratteri accentati vengono gestiti correttamente.

Per poter creare correttamente l'indice degli autori con *MakeIndex* o xindy è necessario che i nomi vengano inseriti come valore della chiave `by` usando la sola virgola come separatore.

4 www.canzoniereonline.it

La classe `canzoniereonline.cls` sta alla base di un sistema per la creazione di canzonieri tramite internet. Chiunque potrà collegarsi al sito www.canzoniereonline.it e ottenere, dopo alcune semplici operazioni, un pdf pronto per la stampa o la consultazione da pc.

Dal punto di vista della programmazione, l'applicativo web presenta alcune sfide interessanti. Il frontend deve presentare all'utente la lista delle canzoni disponibili sul server, consentire una ricerca rapida, fare in modo che l'utente possa scegliere le canzoni che vuole inserire nel canzoniere, le ordini come preferisce, ne scelga la tonalità e imposti i parametri della formattazione. Il backend deve raccogliere le informazioni inserite dall'utente, ricavarne un file latex unico compilabile, compilarlo e restituire il file pdf.

Dopo un'analisi delle possibili soluzioni si è scelto di utilizzare il framework *laravel* per la gestione del backend, appoggiandosi quindi al linguaggio PHP e ad un database mysql per lo storage delle canzoni. Il frontend è invece basato su *bootstrap* per la parte di presentazione grafica e angular per quella di applicativo online.

Lo schema logico dell'applicazione è il seguente:

- le canzoni sono tutte salvate in una tabella del database che contiene sia il `.tex` completo che tutti i metadati (titolo, autore, tonalità, momento, ecc...);
- quando l'utente si collega alla pagina, viene caricato un elenco delle canzoni;
- angular.js gestisce lato client la ricerca delle canzoni, l'aggiunta e l'ordinamento delle stesse nonché la scelta delle tonalità;
- quando l'utente ha scelto le canzoni e inserito i dati richiesti (formattazione e indirizzo email) invia i dati al server;
- il server non procede immediatamente con la creazione del file e la compilazione, ma aggiunge le informazioni a una coda di esecuzione;
- l'utente viene informato che a breve riceverà una mail con il canzoniere;
- prendendo una richiesta alla volta, il server crea un file `.tex` con le informazioni date dall'utente (recuperando dal database il `.tex` delle canzoni), ne compila il pdf in una cartella privata e lo invia all'utente attraverso il servizio Mandrill.

Le prossime implementazioni prevedono la possibilità di consultare le singole canzoni, sia in pdf che in formato HTML, la creazione di un CMS di supporto al sito e un sistema di registrazione per consentire agli utenti di salvare e condividere i loro lavori.

Purtroppo al momento della chiusura di questo articolo l'interfaccia grafica del sito non è ancora completata, pertanto non ci è stato possibile aggiungere delle immagini che illustrino passo passo la procedura per la creazione di un canzoniere.

- ▷ Francesco Endrici
Trento
francescoendrici+tex@gmail.com
- ▷ Matteo Conci
Trento
concimatteo@gmail.com

Creare un database bibliografico da un file di testo.

Una proposta

Matteo Fadini

Sommario

L'articolo esamina una possibile procedura per creare un database bibliografico (`.bib`) a partire da una bibliografia presente in un file di testo ed è una rielaborazione della soluzione proposta da Beatrice Penati (PENATI).

Abstract

This article discusses the possible steps to create a bibliographic database (`.bib`) from a bibliography included in a text file, by reworking the former procedure proposed by Beatrice Penati (PENATI).

1 Introduzione

Esistono diversi strumenti per convertire in $\text{\LaTeX}$ un testo scritto con un programma di videoscrittura; ad esempio *rtf2latex2e*,¹ il plugin per Open/LibreOffice *writer2latex*² oppure il programma a pagamento *word2latex*.³ Ciò che manca a questi programmi è la capacità di convertire efficacemente le citazioni bibliografiche e gli elenchi bibliografici in un file `.bib` correttamente formato, prerequisito fondamentale per poter gestire la bibliografia tramite $\text{\LaTeX}$ e Bib $\text{\TeX}$ /Biber.

Nel corso del lavoro di pubblicazione di una neonata rivista scientifica di ambito letterario,⁴ l'ostacolo maggiore è stato la conversione delle bibliografie presenti nei singoli articoli, tutti presentati sotto forma di file di testo (`.doc`, `.rtf` ecc.).

Tralasciando le ipotesi di comporre a mano *ex novo* il database bibliografico e di copiare e incollare ogni singola referenza nel file `.bib`, è stato ideato il seguente sistema, che necessita di alcuni passaggi e di un file testuale opportunamente formato, ma che risulta una buona soluzione nel caso di bibliografie molto corpose.

La soluzione prospettata riguarda la conversione di un elenco bibliografico in formato testuale in un file `.bib`; nell'articolo si tralascia di fornire indicazioni generali sulla gestione della bibliografia con $\text{\LaTeX}$ (per una guida introduttiva si rinvia al capitolo 7 di PANTIERI e GORDINI (2012) e alla parte

IV di PANTIERI (2009) per alcune informazioni aggiuntive).

2 Premessa

Il procedimento illustrato offre un notevole risparmio di tempo a patto che l'elenco bibliografico abbia una certa estensione, diciamo superiore alla ventina di *item*, e che contenga citazioni non eccessivamente complicate riguardanti la tipologia `book` e `article`: la presenza di indicazioni di traduzioni o prime edizioni originali, come pure le citazioni di miscellanee, non sono facilmente adattabili alla soluzione qui presentata.

3 Preparativi

Occorre anzitutto disporre di un editor di testo qualsiasi e di una versione di (Open/Libre)Office.⁵

Secondariamente, bisogna aver cura di suddividere il file di testo che presenta la bibliografia⁶ in due file separati (`bibliografia-libri` e `bibliografia-articoli`) contenenti, rispettivamente, l'elenco dei volumi e dei contributi apparsi su rivista. Per comodità di esposizione, si prenda come esempio del contenuto del primo file il testo presente nell'appendice A (si veda p. 3) e come esempio del secondo file il testo dell'appendice B (si veda p. 4).⁷

Si formattino in modo uniforme i due file: si controlli che tutti i nomi degli autori siano presentati in maniera identica ('Nome Cognome' oppure 'Cognome, Nome') e che le varie parti della citazione bibliografica siano sempre nella medesima forma e successione (es. Autore, Titolo, Luogo, Casa editrice, anno per i libri; Autore, Titolo, nome rivista, volume, anno, pagine per gli articoli). Si eliminino tutti gli spazi dopo le virgole e le virgolette caporali/alte dai titoli delle riviste tramite il comando 'cerca e sostituisci', così come tutte le indicazioni 'p.', 'pp.' e 'in'; si sostituiscano con un trattino basso (`_`) tutte le eventuali virgole presenti all'interno dei titoli e sottotitoli e si pospongano

5. Benché sia possibile utilizzare la versione libera del software, nel corso dell'articolo verrà trattato MS Word per la semplice ragione che l'utilizzo della funzione di 'Stampa unione' – passaggio necessario al processo – è più semplice con questa versione del programma.

6. Qui e nel prosieguo, con il termine *bibliografia* si intende la lista delle opere citate.

7. Gli elenchi bibliografici sono tratti dal sito del progetto LICAPV - *Libri CAvallereschi in Prosa e in Versi* dell'Università di Pavia (LICAPV).

1. <http://rtf2latex2e.sourceforge.net>.

2. <http://writer2latex.sourceforge.net>.

3. <http://www.wordtolatex.com>.

4. La rivista in questione è «Ticontre. Teoria Testo Traduzione», liberamente consultabile online all'indirizzo www.ticontre.org.

infine a dopo l'anno di pubblicazione le eventuali indicazioni di curatele e tutte le altre particolarità; nel caso di opere senza autore e, più in generale, di campi mancanti, si inserisca una virgola nel luogo mancante.

Il risultato delle operazioni sui due file è rappresentato dalle immagini 1 e 2.

Andrea da Barberino, Aspramonte, Bologna, Antiquaria Palmaverde, 1951, a cura di Marco Boni
Matteo Maria Boiardo, Orlando innamorato, Torino, Einaudi, 1995, a cura di Riccardo Brusca
. Cantari di Aspramonte inediti (Magl. VII 682), Bologna, Commissione per i testi di lingua, 1981 a cura di Andrea Fassò,
. Catalogue of Books printed in the XV Century now in the British Museum, London, British Museum, 1963
Tanmaro De Marinis, Appunti e ricerche bibliografiche, Milano, Hoepli, 1940
Gaetano Melzi, Bibliografia dei romanzi cavallereschi italiani, Milano, Tosi, 1838
Carlo Enrico Rava, Supplement à M. Sander, Le livre à figures italiens de la Renaissance, Milano, Hoepli, 1969.
Dennis E. Rhodes, Gli annali tipografici fiorentini del XV secolo, Firenze, Olschki, 1988.
Max Sander, Le livre à figures italiens depuis 1467 jusqu'à 1530. Essai de sa bibliographie et son histoire, Milano, Hoepli, 1942.
Arnaldo Segarizi, Bibliografia delle Stampe popolari italiane della R. Biblioteca Nazionale di S. Marco di Venezia, Bergamo, Istituto Italiano d'Arti Grafiche, 1913.

FIGURA 1: Schermata del file `bibliografia-libri` formattato come detto sopra

Umberto Cianciolo, Materia leggendaria e giullaresca nel cantare di San Giusto Paladino, Archivum romanicum, XIX, 1935, 183-211.
Umberto Cianciolo, Contributo allo studio dei cantari di argomento sacro, Archivum romanicum, XXII, 1938, 163-84.
Carlo Dionisotti, Appunti su cantari e romanzi, Italia medioevale e umanistica, XXXII, 1989, 227-61.
Lamberto Donati, Un'edizione quattrocentesca del Libro della Regina Ancoira, La Bibliofilia, XLVII, 1945, 40.
Neil Harris, The 'Unicum' of the Second Edition of Boiardo's Orlando innamorato and a Forgery of the Last Century, Rivista di letteratura italiana, IV, 1986, 519-36.
Neil Harris, Il Guernio o l'Ancoira a scelta in una silografia quattrocentesca, La Bibliofilia, XCI, 1989, 95-100.
Stefan Hartung, Komplexes Spiel mit epischer Tradition und die Abwertung der Karlsfigur im Innamoramento di Carlo Magno (1481): Eine Widerlegung der Fusionstheorie der Boiardo Forschung, Romanische Forschungen, CIII, 1991, 172-210.
Martin Lowry, Nel Beretino Convento: The Franciscans and the Venetian Press (1474-78), La Bibliofilia, LXXXV, 1983, 27-40.
Elio Melli, Nella selva dei Rinaldi: poemetti su Rinaldo da Mont'Albano in antiche edizioni a stampa, Studi e problemi di critica testuale, XVI, 1978, 193-215.
Cristina Montagnani, Per l'edizione dell'Orlando Innamorato: una premessa linguistica, Studi di filologia italiana, XLVI, 1988, 131-61.

FIGURA 2: Schermata del file `bibliografia-articoli` formattato come detto sopra

3.1 Creazione del file csv

Dopo aver predisposto in maniera opportuna i due file come appena spiegato, li si salvano come testo (.txt, con codifica UTF 8; ciò comporterà l'eliminazione di tutta la formattazione) e, successivamente, si rinominano i file con l'estensione .csv. A questo punto si possono aprire i file con un programma quale MS Excel o Open/LibreOffice Calc (si consiglia la seconda ipotesi, poiché la versione libera gestisce in maniera migliore la codifica UTF8).

3.1.1 bibliografia-libri.csv

Si apra il file `bibliografia-libri.csv` con uno dei programmi appena nominati. Se i nomi degli autori sono presenti nella forma "Nome Cognome" si aggiungano due colonne all'immediata destra di quella contenente i nomi, si evidenzino la colonna contenente i nomi e poi si segua il percorso Dati→Strumenti dati→Testo in colonne. L'operazione porta all'inserimento dei nomi e dei cognomi nelle due colonne a destra appena create. Per completare l'operazione si elimini la prima colonna contenente i nomi completi.

A questo punto, si inserisca una riga all'inizio del file e si annotino opportunamente le colonne con una legenda (es. nome, cognome, titolo, luogo ecc.). Si inseriscano due nuove colonne a sinistra della prima e si digiti questa formula nella cella B2:

=SE(D2=0;F2&" ":"&H2;D2&" ":"&H2)

laddove la colonna D contiene i cognomi degli autori e la colonna F i luoghi di edizione e H l'anno di stampa; questa formula produrrà come risultato "Cognome:anno", se è presente un cognome, oppure "Luogo:anno", se si tratta di un testo senza responsabilità d'autore. Dopodiché si inserisca nella cella A2 questa formula:

=SOSTITUISCI(B2;" ":";")

che eliminerà gli eventuali spazi all'interno dei cognomi composti. A questo punto si copino le due formule e le si incolli in tutte le celle della colonna. Si aggiunga una ulteriore colonna a sinistra chiamandola "key", si copi il contenuto della colonna nella quale sono state inserite le formule anzidette e si incolli in maniera speciale il contenuto (tasto destro→Incolla speciale→Valori). Fatto ciò, si eliminino la colonna contenente le formule e si salvi il file, mantenendo sempre il formato .csv.

A questo punto il file avrà l'aspetto illustrato dall'immagine 3.

	A	B	C	D	E	F	G	H
1	key	nome	cognome	titolo	luogo	edizione	anno	nota
2	Andrea da Barberino	1951	Andrea da Barberino	Aspramonte	Bologna	Antiquaria Palmaverde	1951	a cura di Marco Boni
3	Boiardo	1995	Matteo Maria Boiardo	Orlando innamorato	Torino	Einaudi	1995	a cura di Riccardo Brusca
4	Boiardo	1981	Matteo Maria Boiardo	Cantari di Aspramonte inediti (Magl. VII 682)	Bologna	Commissione per i testi di lingua	1981	a cura di Andrea Fassò
5	London	1963		Catalogue of Books printed in the XV Century	London	British Museum	1963	
6	De Marinis	1940	Tanmaro De Marinis	Appunti e ricerche bibliografiche	Milano	Hoepli	1940	
7	Melzi	1838	Gaetano Melzi	Bibliografia dei romanzi cavallereschi italiani	Milano	Tosi	1838	
8	Rava	1969	Carlo Enrico Rava	Supplement à M. Sander, Le livre à figures italiens de la Renaissance	Paris	Librairie de la Sorbonne	1969	
9	Rhodes	1988	Dennis E. Rhodes	Gli annali tipografici fiorentini del XV secolo	Firenze	Olschki	1988	
10	Sander	1942	Max Sander	Le livre à figures italiens depuis 1467 jusqu'à 1530	Milano	Hoepli	1942	
11	Segarizi	1913	Arnaldo Segarizi	Bibliografia delle Stampe popolari italiane della R. Biblioteca Nazionale di S. Marco di Venezia	Bergamo	Istituto Italiano d'Arti Grafiche	1913	

FIGURA 3: Schermata del file `bibliografia-libri.csv` al termine delle operazioni

3.1.2 bibliografia-articoli.csv

Quanto descritto riguarda ovviamente le citazioni di libri; nel caso degli articoli occorrerà prevedere altre colonne (es. "nome rivista", "volume", "pagine" ecc.) e il risultato sarà quello illustrato dall'immagine 4.

	A	B	C	D	E	F	G	H
1	key	nome	cognome	titolo	rivista	numero	anno	pagine
2	Cianciolo	1935	Umberto Cianciolo	Materia leggendaria e giullaresca nel cantare di San Giusto Paladino	Archivum romanicum	XIX	1935	183-211
3	Cianciolo	1938	Umberto Cianciolo	Contributo allo studio dei cantari di argomento sacro	Archivum romanicum	XXII	1938	163-84
4	Dionisotti	1989	Carlo Dionisotti	Appunti su cantari e romanzi	Italia medioevale e umanistica	XXXII	1989	227-61
5	Donati	1945	Lamberto Donati	Un'edizione quattrocentesca del Libro della Regina Ancoira	La Bibliofilia	XLVII	1945	40
6	Harris	1986	Neil Harris	The 'Unicum' of the Second Edition of Boiardo's Orlando innamorato	Rivista di letteratura italiana	IV	1986	519-36
7	Harris	1989	Neil Harris	Il Guernio o l'Ancoira a scelta in una silografia quattrocentesca	La Bibliofilia	XCI	1989	95-100
8	Hartung	1991	Stefan Hartung	Komplexes Spiel mit epischer Tradition und die Abwertung der Karlsfigur im Innamoramento di Carlo Magno	Romanische Forschungen	CIII	1991	172-210
9	Lowry	1983	Martin Lowry	Nel Beretino Convento: The Franciscans and the Venetian Press	La Bibliofilia	LXXXV	1983	27-40
10	Melli	1978	Elio Melli	Nella selva dei Rinaldi: poemetti su Rinaldo da Mont'Albano in antiche edizioni a stampa	Studi e problemi di critica testuale	XVI	1978	193-215
11	Montagnani	1988	Cristina Montagnani	Per l'edizione dell'Orlando Innamorato: una premessa linguistica	Studi di filologia italiana	XLVI	1988	131-61

FIGURA 4: Schermata del file `bibliografia-articoli.csv` al termine delle operazioni

3.2 File bibliografia-libri.bib tramite "stampa unione"

Si crei un file con MS Word contenente questo testo:

```
@book{,
author = {},
title = {},
address = {},
publisher = {},
year = {},
note = {}}
```

A questo punto si segue questo percorso: Lettere→Seleziona destinatari e si seleziona il file .csv appena creato. Con alcuni opportuni passaggi (menù Lettere→inserisci campo unione), si inseriscano i relativi campi nel file di testo (“cognome, nome” in corrispondenza di **author**, “key” in corrispondenza di **@book** ecc.). Si copi e incolli in basso il contenuto del file, inserendo prima di **@book** il campo unione “Next” e si ricopi il contenuto tante volte quanti sono i riferimenti bibliografici.

Cliccando su “Anteprima risultati” si ottiene il file bibliografico correttamente composto: le immagini 5 e 6 mostrano l’aspetto che avrà il file in questione, prima e dopo l’anteprima dei risultati. A questo punto occorre solamente sostituire i trattini bassi con la virgola e inserire (manualmente) i campi **subtitle** e simili laddove opportuno. Si salvi il file come testo (.txt) e lo si rinomini con l’estensione .bib.

Aperto il file così rinominato si ottiene il database bibliografico ricercato.

Anche in questo caso, l’esempio riguarda le citazioni di libri, ma è facilmente adattabile al caso di articoli.

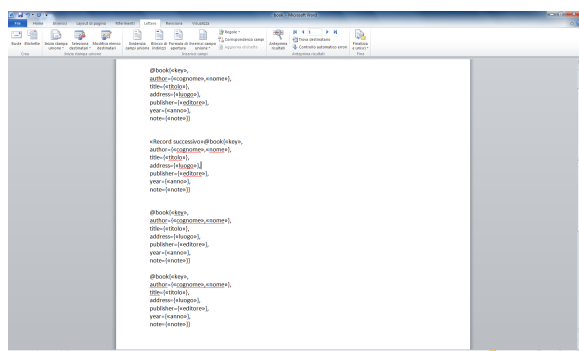


FIGURA 5: Schermata del file word con le formule

4 Conclusioni e limiti

Come si è visto, l’intera procedura consta di numerosi passi che, nonostante siano semplici, richiedono tempo e concentrazione. Questo sistema può essere di una qualche utilità solo a patto di dover trasformare un corposo elenco bibliografico testuale in un database bibliografico e solo se il file che li contiene presenta le varie citazioni in maniera uniforme.

Per esperienza, negli studi umanistico-letterari, la maggior parte dei riferimenti bibliografici è a libri o ad articoli: come già detto, questo sistema

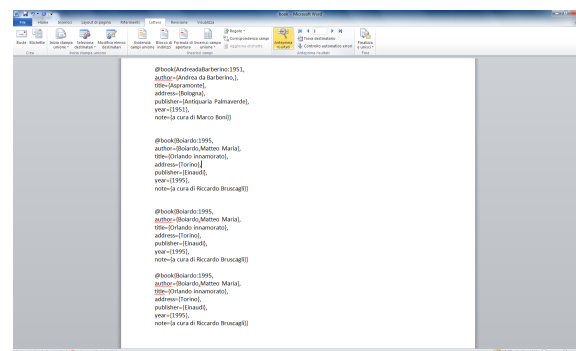


FIGURA 6: Schermata del file word al termine delle operazioni

obbliga a rifare tutta la procedura per ogni tipologia bibliografica e, di conseguenza, è consigliabile solo se la stragrande maggioranza delle citazioni appartiene a non più di tre tipologie.

Riferimenti bibliografici

LICAPV. *Libri CAvallereschi in Prosa e in Versi*.

URL <http://lica.unipv.it/index.php>.

PANTIERI, L. (2009). *Complementi all’arte di scrivere con L^AT_EX*. URL http://www.lorenzopantieri.net/LaTeX_files/Complementi.pdf.

PANTIERI, L. e GORDINI, T. (2012). *L’arte di scrivere con L^AT_EX*. URL http://www.lorenzopantieri.net/LaTeX_files/ArteLaTeX.pdf.

PENATI, B. «From spreadsheet (excel) to .bib file - a simple solution». URL https://www.academia.edu/1747759/From_spreadsheet_Excel_to_.bib_file_-_a_simple_solution.

Appendice A - bibliografia-libri

Andrea da Barberino, *Aspramonte*, a cura di Marco Boni, Bologna, Antiquaria Palmaverde, 1951.

Matteo Maria Boiardo, *Orlando innamorato*, a cura di Riccardo Bruscagli, Torino, Einaudi, 1995.

Cantari di Aspramonte inediti (Magl. VII 682), a cura di Andrea Fassò, Bologna, Commissione per i testi di lingua, 1981.

Catalogue of Books printed in the XV Century now in the British Museum, London, British Museum, 1963.

Tammaro De Marinis, *Appunti e ricerche bibliografiche*, Milano, Hoepli, 1940.

Gaetano Melzi, *Bibliografia dei romanzi cavallereschi italiani*, Milano, Tosi, 1838.

Carlo Enrico Rava, *Supplement à M. Sander Le livre à figures italien de la Renaissance*, Milano, Hoepli, 1969.

Dennis E. Rhodes, *Gli annali tipografici fiorentini del XV secolo*, Firenze, Olschki, 1988.

Max Sander, *Le livre à figures italien depuis 1467 jusqu'à 1530, Essai de sa bibliographie et son histoire*, Milano, Hoepli, 1942.

Arnaldo Segarizzi, *Bibliografia delle Stampe popolari italiane della R. Biblioteca Nazionale di S. Marco di Venezia*, Bergamo, Istituto Italiano d'Arti Grafiche, 1913.

Appendice B - bibliografia-articoli

Umberto Cianciolo, *Materia leggendaria e giularesca nel cantare di San Giusto Paladino*, in «Archivum romanicum», XIX, 1935, pp. 183-211.

Umberto Cianciolo, *Contributo allo studio dei cantari di argomento sacro*, in «Archivum romanicum», XXII, 1938, pp. 163-84.

Carlo Dionisotti, *Appunti su cantari e romanzi*, in «Italia medioevale e umanistica», XXXII, 1989, pp. 227-61.

Lamberto Donati, *Un'edizione quattrocentesca del «Libro della Regina Ancroia»*, in «La Bibliofilia», XLVII, 1945, p. 40.

Neil Harris, *The 'Unicum' of the Second Edition of Boiardo's «Orlando innamorato» and a Forgery of the Last Century*, in «Rivista di letteratura italiana», IV, 1986, pp. 519-36.

Neil Harris, *Il Guerino o l'Ancroia a scelta in una silografia quattrocentesca*, in «La Bibliofilia», XCI, 1989, pp. 95-100.

Stefan Hartung, *Komplexes Spiel mit epischer Tradition und die Abwertung der Karlsfigur im Innamoramento di Carlo Magno (1481): Eine Widerlegung der Fusionstheorie der Boiardo Forschung*, in «Romanische Forschungen», CIII, 1991, pp. 172-210.

Martin Lowry, *«Nel Beretìn Convento»: The Franciscans and the Venetian Press (1474-78)*, in «La Bibliofilia», LXXXV, 1983, pp. 27-40.

Elio Melli, *Nella selva dei «Rinaldi»: poemetti su Rinaldo da Mont'Albano in antiche edizioni a stampa*, in «Studi e problemi di critica testuale», XVI, 1978, pp. 193-215.

Cristina Montagnani, *Per l'edizione dell'«Orlando Innamorato»: una premessa linguistica*, in «Studi di filologia italiana», XLVI, 1988, pp. 131-61.

▷ Matteo Fadini

Università degli studi di Trento | Dipartimento di Lettere e Filosofia

matteo dot fadini at unitn dot it

Zbl-build: a GUI interface for Biblatex

Guido Milanese

Abstract

A set of dialogues and questions helps the user in setting a basic configuration of Biblatex, and in selecting the required BIBTEX archive(s). A more detailed choice is offered for the “philosophy” bundle of styles.

Sommario

Una serie di dialoghi e di domande aiuta l'utilizzatore nella preparazione della configurazione di base per Biblatex nella scelta degli archivi BIBTEX necessari. Per la famiglia di stile “philosophy” viene presentata una maggiore ricchezza di parametri.

1 Why a GUI for Biblatex

Almost ten years ago, in 2006, the first version of Biblatex showed that a new approach to bibliographical issues was possible. BIBTEX, a venerable tool published in 1988, was still the only engine available to L^AT_EX users – the famous and expected “version 1.0” never saw the light of day¹. Biblatex is a very powerful tool², but it is not easy to deal with: even its manual, at more than 260 pages, discourages many users. Googling the Web it is not difficult to find examples and tutorials, and the package itself offers a substantial amount of examples. Particularly in the area of the humanities, where most users are not comfortable neither with writing nor with reading source code, and where most of them are accustomed to a synchronous WYSIWYG approach, even realising the advantages of asynchronous composition and of a real control of one’s writing is far from being easy. Bibliography is likely to be a sort of nightmare in several areas, such as philosophy, philology, or linguistics, where even books or articles written centuries ago may have their *place d’honneur* in a bibliography, because they are still useful. Many scholars save their bibliographic records using pitiful and highly inefficient approaches, generally not strongly structured or even memory-based.

1. See the clear analysis of HUFFLEN (2012). On L^AT_EX and bibliographies, see – in Italian – PANTIERI (2009); in German VOSS (2011); VOSS (2014, 195-230); SCHLOSSER (2014, 71-206). In English I would refer to the manuals I also refer to within Zbl-build: LEHMAN *et al.* (2015) and VALBUSA (2015). The historical sketch of HUFFLEN (2011) is enlightening.

2. See the advanced features described by VALBUSA (2014).

A BIBTEX database can be easily managed and maintained using one of the several GUI(s) available, such as the very popular Jabref³. Users with little or no technical skills are normally comfortable with Jabref and the like, while they would feel uneasy using a text editor such as vim or emacs. Unfortunately, the bibliographic styles are often not easy to deal with; Biblatex is a very powerful tool for the generation of almost any bibliographical style, but the work must be done “by hand”, i.e. studying the manuals and trying to find the most suitable style.

There were some questions posted to T_EX / L^AT_EX user groups asking if a graphical “generator” of Biblatex styles is available⁴ – something similar to the command line tool `makebst`, used to generate the *bst* BIBTEX style files, often combined with merlin master bibliographical style⁵. Zbl-build is a simple graphical interface geared towards making the choice of a Biblatex style less frustrating, setting Biblatex basic features and selecting one or more BIBTEX archives.

2 Basic description

The rather strange name Zbl-build is easily explained: *bl-build* means «Biblatex build», and «z» refers to the GUI programme called *zenity*, used to write the dialogue and selection windows.

2.1 Why Zenity, and why not

The aim of Zbl-build is to be multi-platform: the first choice was therefore a tcl-tk script. But I must frankly admit that, undoubtedly due to my lack of technical ability, I found some aspects of this language quite difficult to control, and I tried other GUI approaches. Given the down-to-earth nature of Zbl-build, and no bells and whistles being required, I took a look at some possible solutions. Zenity, a Gnome GUI tool, looked very promising, and the results within my Linux xubuntu box are excellent. Zenity has been ported to Windows and to OSX, while the excellent program called yad is apparently a Linux only solution. zenity has been ported to OSX both through MacPorts and Homebrew, and the information given by the Wikipedia

3. See <http://jabref.sourceforge.net/>

4. See, e.g., «Is there a WYSIWYG editor for biblatex styles?», <http://tex.stackexchange.com/questions/67248/is-there-a-wysiwyg-editor-for-biblatex-styles>.

5. See <http://www.ctan.org/tex-archive/macros/latex/contrib/custom-bib>. The most recent version of *makebst* is labelled «version 4.1 from 2003/09/0».

page («the homebrew-zenity project has been discontinued») is outdated⁶. Much more promising is the situation of the Windows version; the first porting looks definitely frozen⁷, but a newer porting is now at version 3.6: the maintainer wrote on the 28th of August 2015 that he hopes to update to the most recent version⁸.

A possible alternative is a java porting of *zenity*, called *wenity*⁹, but it does not offer a choice of dialogues as needed by the Zbl-build project. If the Windows and/or OSX versions will not meet the expectations, I think I could move to a different multi-platform approach, such as using the PyZenity library¹⁰ or embedding *zenity* calls into a higher level scripting language such as Snobol¹¹.

3 Zbl-build workflow

The programme displays a set of windows, thus setting basic Biblatex options in this order:

1. options for `csquotes`
2. backend, offering `biber`, `bibtex`, and `bibtex8` as options. `Biber` is given as default choice
3. basic style family: `authordate`, `numeric`, `philosophy`, `reading`, `verbose`, `debug`, `draft`. These options are all the basic Biblatex style families plus the `philosophy` style. No default choice.
4. a sub-style selection matching the previous choice. For example, the style *numeric* offers `numeric`, `numeric-comp`, `numeric-verb`, `alphabetic`, `alphabetic-verb`.
5. Sorting order: `nty`, `nyt`, `nyvt`, `anyt`, `anyvt`, `ynt`, `ydnt`, `none`, `debug`. `Nty` is given as default choice
6. Optional additional options: `Isbn`, `Url`, `DOI`, `Eprint` (yes/no choice)
7. Additional options for the chosen family

6. See <https://wiki.gnome.org/action/show/Projects/Zenity> (*zenity* homepage); the homebrew porting was updated in 2014: see <https://github.com/Homebrew/homebrew/commit/90e35255a9e22f7236a86ca3d4820e3b845577e2>. It needs a re-installation of `gnome-doc-utils`: see <https://github.com/Homebrew/homebrew/issues/33680>. Wikipedia wrongly gives the project as discontinued (<https://en.wikipedia.org/wiki/Zenity>).

7. See <http://www.placella.com/software/zenity/>. No hope to receive information: comments have been disabled and the most recent comment was written on the 31st of March 2013.

8. See <http://opensourcepack.blogspot.it/2013/02/zenity-360-portable.html>

9. See <http://kksu.eu/wenity.html>.

10. See http://brianramos.com/?page_id=38.

11. See e.g. my `copac-clean` script (<https://www.ctan.org/pkg/copac-clean>, using Snobol with `tcl-tk`). On Snobol/Spitbol see <http://www.snobol4.org/>. Both the Snobol and the Spitbolx64 programming languages are updated to 2015.

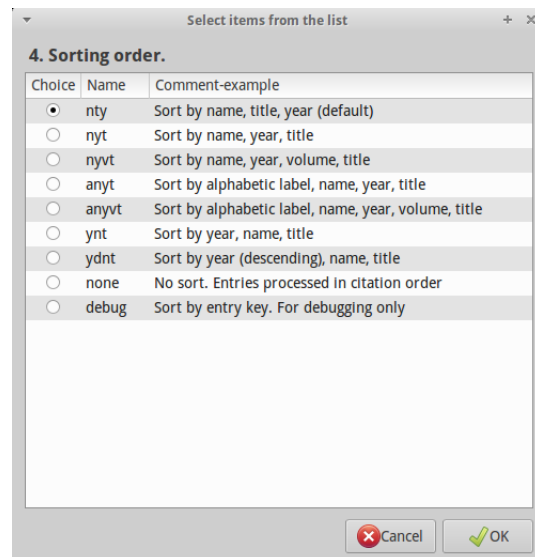


FIGURE 1: The Sorting Order window

8. Selection of one or more BibTeX archives.
9. Finally, an editable window displays the configuration and the user can easily copy/paste it to his LaTeX source file, or input the resulting file, called `biblatex.opt`.

3.1 Information provided to user

All the windows show basic information, copied from the Biblatex manual. For example, see figure 1: even not opening the Biblatex manual, the user is able to make sense of the variety of sorting orders. The default choice is *nty* because it is default in the Biblatex manual (LEHMAN *et al.*, 2015, 45).

The basic style families are the ones provided by the Biblatex manual (LEHMAN *et al.*, 2015, 63 ff.). All the basic style families are provided, but *numeric* is not given as default, because in many areas this is not the case. See also section 4.

The standard style options described by the manual at page 55 (3.1.2.2) are offered by a yes/no window, offering a true/false alternative. Style specific options are offered only for the *philosophy* styles: for all the other styles a message informs that further developments are planned for the forthcoming versions of Zbl-build (see also 4.2). Figure 6 shows the final editing window, where the user can add fine tuning to his configuration,

3.2 The “philosophy” styles

The choice of offering style specific options only for the *philosophy* styles was motivated by the maturity of Ivan Valbusa’s *philosophy* package, but also because it is a sort of killer application in the field of bibliography management for the humanities. A “warning” window (see figure 2) informs the reader that only the basic options are set up by Zbl-build, thus encouraging him to study the manual if not

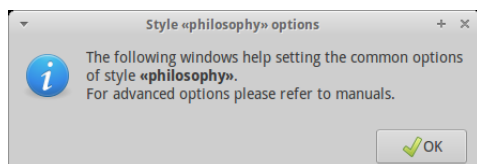


FIGURE 2: A “warning” window

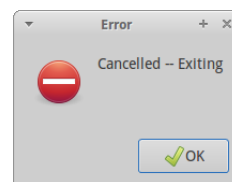


FIGURE 5: The “Cancelled-Exiting” window

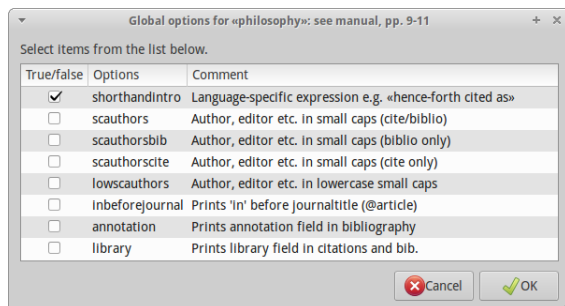


FIGURE 3: Global options for the “philosophy” bundle

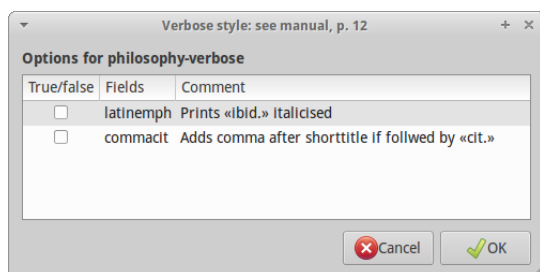


FIGURE 4: Options for the “philosophy-verbose” style

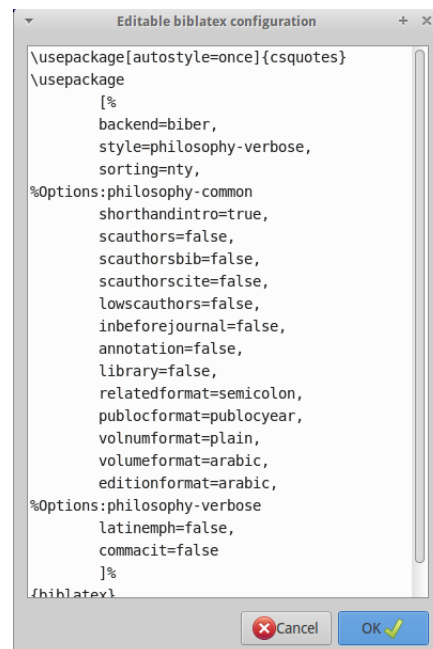


FIGURE 6: The configuration editing window

completely satisfied with the results. Figure 3 shows the options common to all the styles of the bundle; figure 4 shows the options of the *philosophy-verbose* style. In both of the windows, please notice the reference to Valbusa’s manual in the title – again an anti-laziness procedure. Needless to say, most of the refinements offered by the *philosophy* styles are not yet achievable through Zbl-build.

4 Todo

I would like to split the *todo* section into urgent matter, refinements, and doubts.

4.1 Urgent matter

At the moment, the user is offered the option to leave the programme, and if he confirms a message informs that he had actually closed it (see fig. 5.). It would be important to leave the user an opportunity to go back and review his choices.

4.2 Refinements

1. Add for all the styles that support further options the choice given for *philosophy* (see section 3.1).
2. Add support at least for the *chicago* bundle.
3. Support the *natbib* option of Biblatex.

4.3 Doubts

1. One substantial doubt is zenity itself. Will this programme be really usable under OSX? Will the new Windows porting be developed in the foreseeable future?
2. Remove from the configuration all the default choices whenever they take the form of a true/false choice? I’m not sure, because the configuration would be shorter, but an explicit list makes it more helpful.
3. Give to the user the choice of distinguishing citation style and bibliographic style? Or is this option too rarely used to be interesting?

References

- HUFFLEN, J.-M. (2011). «A comparative study of methods for bibliographies». *TUGboat*, **32** (3), pp. 289–301. URL <https://www.tug.org/TUGboat/tb32-3/tb102hufflen.pdf>.
- (2012). «Beyond BibTEX». *ArsTEXnica*, (14), pp. 7–14. URL <http://www.guitex.org/home/numero-14>.
- LEHMAN, P., KIME, P., BORUVKA, A. e WRIGHT, J. (2015). *The Biblatex Package. Programmable*

- Bibliographies and Citations. Version 3.0.*
URL <http://ctan.org/CTAN/macros/latex/contrib/biblatex/doc/biblatex.pdf>.
- PANTIERI, L. (2009). «L'arte di gestire la bibliografia con biblatex». *ArsT_EXnica*, (8), pp. 48–60. URL <http://www.guitex.org/home/numero-8>.
- SCHLOSSER, J. (2014). *Wissenschaftliche Arbeiten schreiben mit LaTeX: Leitfaden für Einsteiger*. mitp Verlag, Heidelberg, 5^a edizione. URL <https://books.google.it/books?id=CmCUAgAAQBAJ>.
- VALBUSA, I. (2014). «Funzionalità avanzate del sistema biblatex/biber». *ArsT_EXnica*, (18), pp. 70–80. URL <http://www.guitex.org/home/numero-18>.
- (2015). *The biblatex-philosophy bundle v1.1*. URL <http://www.ctan.org/pkg/biblatex-philosophy>.
- VOSS, H. (2011). *Bibliografien mit L^AT_EX*. Lehmanns Media, Berlin. URL <https://books.google.it/books?id=SgNACgAAQBAJ>.
- (2014). *L^AT_EX-Referenz*. Lehmanns Media, 3^a edizione. URL <https://books.google.it/books?id=OMQsCgAAQBAJ>.
- ▷ Guido Milanese
Università Cattolica del Sacro Cuore, Milano–Brescia
guido dot milanese at unicatt dot it

MLBibT_EX 1.4: the New Version

Jean-Michel Huppen

Abstract

We present the features of MLBibT_EX 1.4. This version has been especially designed to deal with character encodings. It also provides more constructs allowing users to customise the services of this bibliography processor.

Keywords MLBibT_EX, bibliography processor, Unicode, character encodings, interface with Scheme.

Sommario

Presentiamo le funzionalità di MLBibT_EX 1.4. Questa versione è stata progettata apposta per gestire le codifiche dei caratteri. Fornisce anche più strutture che permettono agli utenti di personalizzare le funzioni di questo elaboratore di bibliografie.

Parole chiave MLBibT_EX, elaboratore di bibliografie, Unicode, codifiche dei caratteri, interfaccia con Scheme.

Preliminary note The bibliography of the present article has been processed with MLBibT_EX, as a showcase for this program.

1 Introduction

It is currently well-known that typesetting systems built out of T_EX—i.e., *Plain T_EX*, L^AT_EX (Mittelbach and Goossens, 2004) or ConT_EXt (Hagen, 2001)—provide unrivalled high-quality outputs. In addition, T_EX itself provides a programming language so that new commands can be defined. Often, such commands are used for frequently performed tasks, but also for less frequent complex typesetting procedures. A good example of this second case is given by the commands and environments of the *wrapfig* package (Mittelbach and Goossens, 2004, § 6.4.1), allowing text to be wrapped around figures. Despite T_EX language is very suitable for handling a stream of characters and even though *any* procedure could be programmed with T_EX¹, implementing those operations more related to ‘pure’ programming, e.g., a sort procedure (van der Laan, 1993), may be tedious. At first glance, such a procedure is not essential for a word processor. However it can be useful if this word processor has

to typeset items extracted from a database and these items are to be sorted. The LuaT_EX engine (Hagen, 2006) allows its users to apply functions written in Lua (Ierusalimsky, 2006) to data before processing them with T_EX. In other words, operations related to typesetting process are delegated to T_EX, whereas functions belonging to ‘pure’ programming can be written in Lua.

There is some analogous evolution about bibliography processors, in the sense that programming features have become more and more accessible. Let us recall that a bibliography processor searches bibliography database files for resources associated with bibliographical keys cited throughout a document, and allows a word processor to typeset these resources. More often, they are grouped into a ‘References’ section put at the end of a document; sometimes, they are given in footnotes instead of in-text citations. Let us consider BibT_EX (Mittelbach and Goossens, 2004, Ch. 13), the bibliography processor ‘traditionally’ used with L^AT_EX: it provides a language for designing *bibliography styles*, ruling the layout of a references section. It is easy to introduce small changes into an existing BibT_EX bibliography style—e.g., using italicised characters for any title—as shown in Mittelbach and Goossens (2004, § 13.6.3). BibT_EX uses a bibliography style language which, although old-fashioned, is very suitable to organise successive text fragments, but makes hard, or even impossible, to sort bibliography items according to several cascading sort keys. A more recent *modus operandi* has been put into action by the *biblatex* package (Lehman, 2014): files for bibliographical references (.bbl files) are marked up with L^AT_EX commands, and the choice of a bibliography layout is deferred to the processing of such a file by L^AT_EX. In other words, typesetting a bibliography can take advantage of L^AT_EX’s full expressive power². ConT_EXt users can use the same tool thanks to the *bib* module (CONTEXTGARDEN, 2012). Consider now MLBibT_EX³, our reimplementation of BibT_EX written in Scheme (Kelsey et al., 1998)—a functional programming language: it can be used “as-is” by non-computer scientists. In addition, some difficult or impossible operations within BibT_EX can be easily put

1. From a theoretical point of view, T_EX’s language has been proved Turing-complete. Interested readers can find a didactic presentation of this property at Belmans (2010). In practice, this means that any *computable* function can be programmed with such a language.

2. Now, the *biblatex* package is often used with the *biber* bibliography processor (Kime and Charette, 2014). Since *biber* can only be used with *biblatex*, it does not provide any way to implement bibliography styles and such an action is actually deferred to L^AT_EX.

3. MultiLingual BibT_EX.

into action within MIBibTeX by customising some Scheme definitions. That is, a bibliography processor should be used by non-computer scientists, but some operations, related to pure programming, could be expressed using a more expressive programming language. As we will show, this point will be strengthened for the new version.

In the following section, we will summarize the history of MIBibTeX's successive versions, some of the experience we got, and the requirements of next version. Features that will be introduced in the next version are described in Section 3 and changes concerning the commands related to MIBibTeX are shortly given in Section 4.

2 MIBibTeX: past, present and future

2.1 Previous versions

Version 1.1 was distributed only confidentially⁴, Version 1.2—experimental—was abandoned, and for a long period of time, the present version (1.3) (Huffle, 2003) has incorporated many releases—identified by geographical names—enlarging and improving it⁵. Moreover, some new functions overstepped the production of bibliographical references for documents processed by a typesetting system like L^AT_EX. As an example, MIBibTeX's kernel has been successfully used to populate an open archive repository from publications described in bibliography database files (.bib files) (Huffle, 2010).

2.2 Bridges towards other tools

MIBibTeX was first designed to work with L^AT_EX, and then adapted to ConT_EXt, with some features allowing ConT_EXt-specific definitions to be processed. As a feature originating from the bibl_{at}ex package, MIBibTeX can deal with the DATE field, even if 'classical' bibliography styles are used⁶. Let us go on with bibl_{at}ex; some additional bibliography types recognised by this package can now be processed by MIBibTeX, e.g., the @BOOKINBOOK type. MIBibTeX does not replace biber but allows most extensions recognised by biber and bibl_{at}ex to be used within other contexts.

2.3 Recent requirements

As explained in Mittelbach and Goossens (2004, Ch. 13), BibTeX was originally designed to work

with ASCII⁷ characters. This program was refined into BibTeX8 in order to deal with some encodings, such as Latin 1 or Latin 2, but according to an *ad hoc* way. MIBibTeX's present version internally uses the Latin 1 encoding, and can read .bib files according to this encoding. Roughly speaking, this is not too restrictive, because letters and signs outside this encoding are recognised by a kind of normalisation of the L^AT_EX commands used to produce them, as explained in Huffle (2015). These 'normalised' commands are recognised by our functions, in particular by our language-dependent order relations used to sort bibliographies. Another point is that Unicode (The UNICODE CONSORTIUM, 2013) has succeeded and some encodings that cover Unicode's full range are now widely used, e.g., UTF-8. Besides, UTF-8 is used as the main encoding of new engines such as X_YT_EX (Kew, 2007) or LuaT_EX. When we started implementing MIBibTeX, only a few programming languages were able to deal with the full range of Unicode characters; now it is a drawback for a program not to deal with the UTF-8 encoding.

Another point: a *huge* number of .bib files are available as bibliographies in many domains, in particular on the Web. However, this is a *legacy* format, and other format have appeared. A modern bibliography processor should be able to process .bib files, but also other formats. A good compromise is the ability to process formats based on XML⁹-like syntax.

2.4 Our experience

Any new user of MIBibTeX quickly notices that this program is less permissive than 'old' BibTeX. This choice is deliberate: BibTeX considers *a priori* that any value associated with a bibliographic field—e.g., TITLE—can be *any* string. If such a string is not well-formed for a more structured field such as AUTHOR when BibTeX's bibliography styles are applied, a faulty result may look strange within generated bibliographies or even go unnoticed. Likewise, *any* string is accepted as a value associated with the YEAR field, even if BibTeX expects to process four digits. In particular, this choice allows BibTeX to perform only lexicographic sorts when a bibliography has to be ordered, but processing ancient references—for example, involving negative years, *before Christ*—is impossible¹⁰. On the contrary, MIBibTeX performs strict checking of some data—what may be viewed as a kind of *type-checking*—

4. It was written mainly in C (Kernighan and Ritchie, 1988).

5. MIBibTeX 1.3's last release, incorporating inexact years and authors whose identity is surmised, is 'Verona's' (Huffle, 2014b), because it was developed for the last qJr conference.

6. BibTeX's predefined bibliography styles are based on YEAR and MONTH fields. Of course, MIBibTeX recognises them. It can also deal with a DAY field, and these three informations can be superseded by a DATE field.

7. American Standard Code for Information Interchange.

8. Unicode Transformation Format.

9. eXtensible Markup Language.

10. The same drawbacks exist with biber: within the documentation of these two tools, you can see that a *type* information is associated with each field, but in practice, only warning messages are emitted if ill-formed data are processed. Besides, only lexicographic sorts are performed, like in BibTeX.

e.g., a value associated with the `YEAR` field *must* be a number¹¹. Otherwise, MLBIB_T_EX signals an error and stops immediately. Roughly speaking, this choice causes fatal errors for ill-formed `.bib` files, but allows us to perform *specific* operations, *type-oriented*—for example, *numerical* sorts on years. We personally think that the advantages of this approach supersede its drawbacks. In addition, typos in optional field names can be detected, whereas BIB_T_EX silently ignores such fields.

Putting more checks on bibliographical entries fields into action needs the development of new functions in `Scheme`, but connecting these new functions to the existing ones is quite easy. As an example, let us consider the `mlbibtex2xml` program, generating an XML file from `.bib` files, according to the format we use internally: if the `-publiweb` option is active¹², we check that journal names are included into a given list. Likewise, when authors' names are processed, we search a database of persons, so the names of people included into this database¹³ can be emphasised within a publication list.

Another interesting example, which would have been very difficult to put into action if new `Scheme` functions were not developed, was given by our institute activity report. The publication list of members had to be sorted regarding: 1) research teams, 2) years—in reverse chronological order—, 3) authors' names, 4) months—in reverse chronological order, too. As we showed at a previous Q_JT conference, it is easy to chain sort keys within order relations of MLBIB_T_EX (Hufflen, 2012b).

3 The new version

The major motivation of MLBIB_T_EX's new version (1.4) is the development of a Unicode-compliant version, able to deal with most encodings, such as Latin 1, Latin 2, ... and UTF-8. We judge interesting and very useful some features of the present version (i.e., the type-checking for values associated to some fields) or the use of a more expressive programming language for operations related to pure programming (i.e., the definition of advanced sort procedures). So, MLBIB_T_EX 1.4 has the same architecture of MLBIB_T_EX 1.3, but characters may belong to the full range of Unicode. This last point only concerns the values associated with fields such as `AUTHOR`, `TITLE`, etc. Characters allowed within field names and bibliographical keys still follow the old rules¹⁴.

11. ... or possibly an *inexact* number, as shown in Hufflen (2014b).

12. This option has been put into action upon request of people in charge of publications within our institute.

13. That is, our institute's members.

14. That is, accented letters will not be permitted within these cases.

```
%encoding = latin1
@BOOK{houssin-vila1976,
  EDITOR = {Joël Houssin with
    first => Christian,
    last => Vilà},
  TITLE = {Banlieues rouges},
  SERIES = {Nébula},
  NUMBER = 12,
  PUBLISHER = {Opta},
  TOTALPAGES = 240,
  YEAR = 1976,
  MONTH = sep,
  NOTE = {[Made headlines in the middle
    1970's] ! english},
  LANGUAGE = french}
```

FIGURE 1: A `.bib` file using MLBIB_T_EX's extensions.

3.1 Directives

MLBIB_T_EX 1.3's latest releases recognise some *directives*—starting with the `'%` character and put at the beginning of `.bib` files—that give a namespace for bibliographical keys (Hufflen, 2014a) and specify a `.bib` file character encoding if it cannot be guessed. Within these releases, only ASCII and Latin-1 are processed. MLBIB_T_EX 1.4's first release will be able to provide Unicode encodings byte-based—that is, Latin-1, Latin-2, UTF-8, but not UTF-16, which will be available in the near future. The encoding of a generated bibliography could be chosen; besides, MLBIB_T_EX will be able to use *several* encodings for parsed bibliography databases: for example, Latin-1 for a first `.bib` file, Latin-2 and UTF-8 for a second and third ones, etc. The default encoding will be still Latin-1, except for some specific cases, e.g. `mlbibcontext`'s output encoding (UTF-8).

A short example using the `'%encoding'` directive is given in Figure 1; it also gives some syntactic extensions used by MLBIB_T_EX for personal names and multilingual annotations (see Hufflen (2003) for more details).

3.2 More formats

MLBIB_T_EX 1.4 will be able to read bibliography data as XML files according to the rules sketched in Hufflen (2002). More precisely, let *f* be a filename for bibliographical entries, given without suffix. MLBIB_T_EX 1.4 first will look for *f.bib*, second for *f-mlbiblio.xml*.

3.3 Interface with Scheme

As above mentioned (cf. § 2.4), MLBIB_T_EX 1.3 makes it possible to add new definitions in `Scheme` and improve the code quality of some functions, but in practice this ability was only used by people able to recompile the whole program. MLBIB_T_EX 1.4 will look for initialisation files in users' home direc-

tories. These Scheme files could add new definitions or redefine some default conventions. For example, the default encoding for .bbl files generated by MIBibTeX 1.4 is Latin 1. The same for the default input encoding for .bib files. If you would rather prefer UTF-8 instead of Latin 1, just put the following Scheme expressions in your initialisation file:

```
((encodings-pv 'set-default-4-bib-files)
'utf-8)
((encodings-pv 'set-default-4-output)
'utf-8)
```

4 Programs around MIBibTeX

The new arguments and options conventions of the executable programs derived from MIBibTeX's source files are:

- the `mlbibtex` program has an additional option `-outputenc` for the encoding of the .bbl file generated:

```
mlbibtex -outputenc encoding \
job-name
```

the default output encoding is used in case of a missing option;

- so does the `hal` program, allowing an open-archive site to be populated (Huppen, 2010);
- the program `mlbiblatex`'s last argument is for the encoding of the generated .bbl:

```
mlbiblatex job-name \
order-information language-code \
encoding
```

the default output encoding will be used if the argument is missing; let us recall that this program generates .bbl files suitable for the `biblatex` package, see Huppen (2012a) about other arguments;

- there is no change about the arguments of the `mlbibcontext` program:

```
mlbibcontext job-name \
order-information language-code
```

but the encoding of .bbl files generated is always UTF-8; see Huppen (2012a) about the other arguments of this program generating .bbl files suitable for ConTeXt's `bib` module;

- the `mlbibtex2xml` command:

```
mlbibtex2xml filename0 \
[filename1 ...]
```

now accepts two new options¹⁵:

15. The `-publiweb` option, of this program, sketched at § 2.4, is still available.

- `-outputenc encoding` which works like the namesake command of the programs `mlbibtex` and `mlbibtex2xml`;
- `-check` in which case the `mlbibtex2xml` programs only performs syntactic checking about the bibliographical items of .bib files, no XML file is generated; this option is incompatible with the `-o` option.

Let us recall that:

- the `-inexact` option—see Huppen (2014b)—is usable by the programs `mlbibtex` and `mlbibtex2xml`;
- all these programs can use the `-h` or `--help` option.

Each of these programs runs an invisible namesake file, located at the program user's home directory. As mentioned in § 3.3, it is a Scheme source file, used to redefine some default conventions. If we consider Linux or Mac OS X conventions, this file is `~/.mlbibtex` for the executable program `mlbibtex`, `~/.mlbiblatex` for `mlbiblatex`, etc. If such a file is missing, only options can supersede default conventions.

5 Conclusion

From a programming point of view, we think that MIBibTeX's conception has been proved robust, and the resulting program actually maintainable, since we have been able to implement many additional features quite easily. For some years, we waited for a new version of Scheme, with standard predefinitions dealing with Unicode characters, as we explained in Huppen (2012a). Now this new standard has come out (Shinn et al., 2013). Even if developing a Unicode-compliant version implies major rewriting, especially about language-dependent order relations, we are confident about extending and reworking MIBibTeX.

Acknowledgements

I acknowledge Claudio Beccari for his patience while waiting for this article, and for his Italian translation of the abstract and keywords.

References

- Pieter Belmans. *TeX is Turing-Complete*. Universiteit Antwerpen, <https://pbelmans.files.wordpress.com/2010/12/textalk.pdf>, December 2010.
- CONTEXTGARDEN. *Bibliographies in MkIV*. http://wiki.contextgarden.net/Bibliography_mkiv, July 2012.
- Hans Hagen. *ConTeXt, the Manual*. <http://www.pragma-ade.com/general/manuals/cont-enp.pdf>, November 2001.

- Hans Hagen. LuaT_EX: Howling to the moon. *Biuletyn Polskiej Grupy Użytkowników Systemu T_EX*, 23:63–68, April 2006.
- Jean-Michel Huppen. Multilingual features for bibliography programs: From XML to MLBIBT_EX. In *EuroT_EX 2002*, pages 46–59, Bachotek, Poland, April 2002.
- Jean-Michel Huppen. MLBIBT_EX’s version 1.3. *TUGboat*, 24(2):249–262, July 2003.
- Jean-Michel Huppen. Using MLBIBT_EX to populate Open Archives. In Tomasz Przechlewski, Karl Berry, Gaby Gic-Grusza, Ewa Kolsar, and Jerzy B. Ludwichowski, editors, *Typographers and Programmers: Mutual Inspirations. Proc. BachoT_EX 2010 Conference*, pages 45–48, April 2010.
- Jean-Michel Huppen. MLBIBT_EX and its new extensions. In *Proc. 6th ConT_EXt Meeting & EuroT_EX 2012*, pages 82–91, Breskens, The Netherlands, October 2012a.
- Jean-Michel Huppen. Beyond BIBT_EX. *ArsT_EXnica*, 14:7–14, October 2012b. In Proc. GUIT meeting 2012.
- Jean-Michel Huppen. Managing name conflicts and aliasing with MLBIBT_EX. In Tomasz Przechlewski, Karl Berry, Bogusław Jackowski, and Jerzy B. Ludwichowski, editors, *What Can Typography Gain from Electronic Media? Proc. BachoT_EX 2014 conference*, pages 13–16, Bachotek, Poland, April 2014a.
- Jean-Michel Huppen. Dealing with ancient works in bibliographies. *ArsT_EXnica*, 18:81–86, October 2014b. In Proc. GUIT meeting 2014.
- Jean-Michel Huppen. From MLBIBT_EX 1.3 to 1.4. In Tomasz Przechlewski, Karl Berry, Bogusław Jackowski, and Jerzy B. Ludwichowski, editors, *Various Faces of Typography. Proc. BachoT_EX 2015 conference*, pages 13–17, Bachotek, Poland, April 2015.
- Roberto Ierusalimsky. *Programming in Lua*. Lua.org, 2 edition, March 2006.
- Richard Kelsey, William D. Clinger, and Jonathan A. Rees, with Harold Abelson, Norman I. Adams iv, David H. Bartley, Gary Brooks, R. Kent Dybvig, Daniel P. Friedman, Robert Halstead, Chris Hanson, Christopher T. Haynes, Eugene Edmund Kohlbecker, Jr, Donald Oxley, Kent M. Pitman, Guillermo J. Rozas, Guy Lewis Steele, Jr, Gerald Jay Sussman, and Mitchell Wand. Revised⁵ report on the algorithmic language Scheme. *HOSC*, 11(1):7–105, August 1998.
- Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, 2nd edition, 1988.
- Jonathan Kew. X_YT_EX in T_EX live and beyond. *TUGboat*, 29(1):146–150, 2007. EuroBachot_EX 2007 proceedings.
- Philip Kime and François Charette. *biber. A Backend Bibliography Processor for biblatex. Version biber 1.9 (biblatex 2.9)*. <http://ftp.oleane.net/pub/CTAN/biblio/biber/documentation/biber.pdf>, May 2014.
- Philipp Lehman, with Philip Kime, Audrey Boruvka, and Joseph Wright. *The biblatex Package. Programmable Bibliographies and Citations. Version 2.9a*. <http://ctan.mirrorcatalogs.com/macros/latex/contrib/biblatex/doc/biblatex.pdf>, June 2014.
- Frank Mittelbach and Michel Goossens, with Johannes Braams, David Carlisle, Chris A. Rowley, Christine Detig, and Joachim Schrod. *The L_AT_EX Companion*. Addison-Wesley Publishing Company, Reading, Massachusetts, 2 edition, August 2004.
- Alex Shinn, John Cowan, and Arthur A. Gleckler, with Steven Ganz, Aaron W. Hsu, Bradley Lucier, Emmanuel Medernach, Alexey Radul, Jeffrey T. Read, David Rush, Benjamin L. Russel, Olin Shivers, Alaric Snell-Pym, and Gerald Jay Sussman. Revised⁷ Report on the Algorithmic Language Scheme. <http://trac.sacrideo.us/wg/raw-attachmet/wiki/WikiStart/r7rs.pdf>, July 2013.
- The UNICODE CONSORTIUM. *Unicode 6.3.0*. <http://www.unicode.org/versions/Unicode6.3.0/>, November 2013.
- Kees van der Laan. Sorting within T_EX. *TUGBoat*, 14(3):319–328, October 1993.
- ▷ Jean-Michel Huppen
FEMTO-ST (UMR CNRS 6174) &
University of Franche-Comté,
16, route de Gray,
25030 BESANÇON CEDEX
FRANCE
jmhuffle@femto-st.fr

Generare documenti $\text{\LaTeX}$ con diversi linguaggi di programmazione

Roberto Giacomelli, Gianluca Pignalberi

Sommario

Questo articolo mostrerà come generare sorgenti $\text{\LaTeX}$ per mezzo di programmi scritti in diversi linguaggi di programmazione. Vogliamo con ciò presentare una soluzione generale per i progetti di documentazione basati su processi automatici a causa della complessità dei dati o per scelta applicativa, così da ottenere rapporti magnificamente composti.

Abstract

This paper will show how to generate $\text{\LaTeX}$ source code with programs written in different programming languages. We aim at presenting a general solution to those documentation projects based on automatic processes — because of data complexity or on purpose — so to obtain beautifully typeset reports.

1 Introduzione

Non diciamo niente di nuovo rimarcando il fatto che un documento $\text{\LaTeX}$ è un documento di puro testo. La scelta di questa codifica, ridondante dal punto di vista dell'efficiente occupazione di spazio su disco, offre due notevolissimi vantaggi:

1. la possibilità di leggere e modificare ogni documento da *qualunque* editor di testo, interattivo o meno;
2. la possibilità di scrivere programmi e librerie in grado di generare o modificare documenti in maniera automatica e dipendente dall'elaborazione.

Del primo vantaggio non vale la pena discutere oltre, vista la sovrabbondanza di manuali e articoli che elencano questo punto in ogni lista di vantaggi di $\text{\LaTeX}$ rispetto a ogni programma concorrente che salvi in un formato binario.

Intendiamo invece soffermarci sul secondo vantaggio, più che mera estensione del primo. Lo faremo fornendo degli esempi concreti, più o meno semplici, tramite programmi scritti in diversi linguaggi di programmazione e scripting. Ognuno di questi programmi scriverà in parte o completamente un documento $\text{\LaTeX}$ inserendogli i risultati di un'elaborazione. Vedremo diversi metodi: scrittura a partire da un modello — un *template* — con riempimento di parti bianche o con sostituzione di

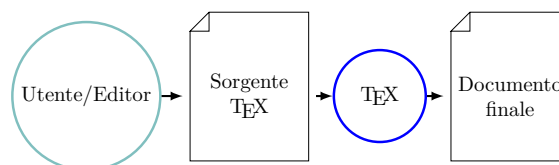


FIGURA 1: Flusso di lavoro manuale. L'utente crea e modifica direttamente un documento $\text{\TeX}$ tramite un editor di testi. Tale documento verrà poi composto dal sistema $\text{\TeX}$.

quelle fittizie; scrittura mediante l'uso una libreria già predisposta, scrittura completa di una tabella mediante scrittura su file di stringhe.

Nel caso di scrittura parziale, lo farà a partire da un modello — un *template* — riempiendo le parti bianche o sostituendo quelle fittizie; nel caso di scrittura totale, scriverà tutto il documento, se necessario inserendo il preambolo.

Dopo aver chiarito il campo applicativo nel paragrafo 2, descriveremo il tipo di dato stringa e le relative operazioni nel paragrafo 3. Il paragrafo 4 introdurrà i modelli in termini generali.

Essendo i paragrafi fin qui elencati a carattere teorico, demanderemo ai successivi paragrafi 6, 7, 8 e 10 di chiarire i dettagli di funzionamento dei comandi specifici del linguaggio in esame e spiegare le tecniche usate con relativi vantaggi e svantaggi. Per ogni linguaggio presenteremo uno o più esempi in grado di fornire una panoramica sufficiente, benché non esaustiva, delle operazioni necessarie a rendere un programma *TeX-friendly*. Arriveremo a tale *TeX-friendliness* mediante programmazione brutta, dunque costruendo delle funzioni *ad hoc*, o per mezzo di librerie preesistenti.

I paragrafi 9 e 11 mostreranno due corposissimi programmi in Lua e Python per risolvere dei problemi reali quali la realizzazione della griglia di attività annuali di un gruppo di persone e un report d'inventario di un esercizio commerciale con interrogazione di database.

2 Il campo applicativo

Il flusso di lavoro normale per generare un documento $\text{\LaTeX}$ prevede che l'utente scriva manualmente i sorgenti usando un editor di testi (figura 1).

Nel caso della produzione automatica della documentazione invece, l'utente non interviene in alcun modo sui sorgenti. Interagisce però con un'applicazione che elabora i dati e ne memorizza il risultato

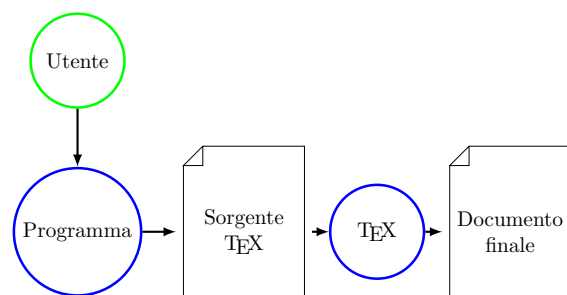


FIGURA 2: Flusso di lavoro automatico. L’utente crea e utilizza un programma che genera il sorgente TEX. Tale documento verrà poi composto dal sistema TEX.

in file perfettamente compilabili da uno dei motori di composizione della famiglia TEX (figura 2).

Stiamo entrando in un mondo completamente diverso dall’usuale: in questa prospettiva considereremo l’utente non come chi scrive il documento ma come chi scrive il programma che scriverà il documento. Un punto di vista che ci possiamo trovare ad affrontare in tutti quei casi in cui i dati del documento finale *richiedano* un’elaborazione automatica iniziale.

Si pensi per esempio alla produzione di report da database: è un campo applicativo dominato da appositi strumenti informatici programmati in un linguaggio ad alto livello da sviluppatori esperti che operano in infrastrutture di rete. Nulla però vieta di esplorare soluzioni basate su TEX perché con questo sistema il report è un testo comprensibile direttamente modificabile dagli utenti.

Il nostro terreno di esplorazione sarà quindi quello che percorrerebbero gli utenti del sistema TEX desiderosi di risolvere in proprio i problemi di reporting per raggiungere due principali obiettivi:

- report di ottima qualità tipografica;
- report che si adattino facilmente a nuove esigenze per contenuto e grafica.

Come vedremo più avanti, in confronto ai linguaggi di programmazione studiati per applicazioni di sistema come il C/C++, i linguaggi di scripting non obbligano a preoccuparsi della gestione della memoria e di molti altri dettagli, a utilizzare i puntatori oppure a usare un compilatore per ottenere gli eseguibili.

In questo articolo mostreremo esempi concreti, didattici o meno, in C e bash e in altri due linguaggi di scripting che giudichiamo importanti per la loro potenziale utilità per gli utenti TEX: Lua e Python.

In particolare, Lua (www.lua.org) ha una sintassi essenziale e una sola struttura dati predefinita, la tabella, che può essere impiegata sia come lista che come dizionario e incorpora una libreria standard che offre all’utente solamente le funzionalità indispensabili.

Il linguaggio Python (www.python.org), invece, si avvale di tre tipi fondamentali di dato (la lista,

la tupla e il dizionario) e presenta una sintassi più vicina al paradigma object oriented. Sono disponibili moltissime librerie, praticamente per tutte le necessità e in particolare per la connessione a database SQLite o PostgreSQL.

Strumenti simili integrati in un flusso di lavoro possono essere le migliori soluzioni possibili in tutti quei casi che potremo definire *artigianali* dove l’utente TEX vuole produrre documenti da insiemi di dati dalle origini e dai formati più diversi.

Progetti in questo senso ne sono già stati realizzati moltissimi: si pensi all’applicazione Facile del comune di Napoli coadiuvata da Agostino De Marco (DE MARCO e CRESCI, 2011) o al recente esempio di produzione di menù per ristorante con un’applicazione web basata su PHP di Claudio Fiandrino (FIANDRINO, 2014). Ma anche la produzione di certificati di prova sui materiali secondo specifiche normative tecniche elaborato da Renato Battistin (BATTISTIN, 2014).

Tutti questi progetti sono recenti perché le esigenze di reporting crescono sempre di più in un mondo connesso e informatizzato come quello delle nostre società. D’altro canto grazie a Internet sono disponibili strumenti software multiplatforma, aperti e ben documentati dalle funzionalità davvero notevoli con cui gli utenti possono impegnarsi per risolvere i problemi di elaborazione dati integrando TEX nel processo.

3 L’elaborazione delle stringhe

La *stringa* è un tipo di dato di *alto livello* che consiste in una sequenza di caratteri per esprimere il concetto di testo. Attraverso le stringhe il testo può essere memorizzato in o caricato da un file (dove per file intendiamo anche il flusso a video o da tastiera e non solo un file su disco).

La locuzione “di alto livello” sta a indicare che tale dato è, di fatto, qualcosa di più complesso: il C memorizza una stringa come un vettore di caratteri gestito per mezzo del puntatore al primo elemento interpretando il carattere NULL (`\0`) come segnale di fine sequenza. Tale terminazione è inserita quando si preme il tasto **invio** — codificato come `\cr\lf`, `\cr` o `\lf` a seconda che il sistema operativo sia DOS/Windows, Mac pre-OS X o Unix — e non fa parte della stringa.

La *stringa* è un tipo di dato di *alto livello* che consiste in una sequenza di caratteri. La locuzione “di alto livello” sta a indicare che tale dato è, di fatto, qualcosa di più complesso: il C memorizza una stringa come un vettore di caratteri (o, meglio, un puntatore a carattere). Il tipo *stringa* esprime dunque esattamente il concetto di contenitore di testo. Il testo può essere memorizzato in una stringa che a sua volta può essere memorizzata, cioè il suo contenuto può essere memorizzato, in un file di testo e viceversa. Naturalmente non c’è una restrizione sul tipo di caratteri contenuti in

una stringa, sebbene il C standard termini una stringa col carattere NULL (`\0`) e tale terminazione venga segnalata premendo `invio`, il cui carattere (`\cr\lf`, `\cr` o `\lf` a seconda che il sistema operativo sia DOS/Windows, Mac pre-OS X o Unix) pertanto non farà parte della stringa.

Per scrivere un programma che generi sorgenti sembra naturale usare le stringhe per tutte le operazioni, ivi compresa la memorizzazione di quelle parti del documento non soggette ad alcuna modifica dipendente dall'elaborazione. L'uso che delle stringhe possiamo fare è in buona misura determinato dalle caratteristiche di questi dati e dalle relative operazioni di manipolazione. Vediamo brevemente le possibili operazioni su una stringa, tenendo conto che ogni linguaggio le implementerà in base alle proprie caratteristiche e con una sintassi specifica, non sempre perfettamente coerente dal punto di vista dell'utente:

creazione: operazione con cui la stringa viene creata, ed eventualmente inizializzata; di fatto le viene riservata un'area di memoria accessibile tramite un'etichetta (il nome della variabile di tipo stringa);

lettura: operazione con cui il programma legge il contenuto della stringa;

scrittura: operazione con cui il programma scrive del contenuto nell'area di memoria riservata alla stringa;

lunghezza: determinazione del numero dei caratteri della stringa;

assegnazione: scrittura del contenuto di una stringa in un'altra stringa;

concatenazione: operazione con cui il contenuto una stringa viene attaccato alla fine del contenuto di un'altra stringa;

comparazione: date due stringhe, se ne confronta il contenuto per determinare quale delle due venga alfabeticamente prima dell'altra;

conversione: una stringa può essere convertita, quando ce ne siano le condizioni, in numero, in lista di token...

interpolazione: espansione di *segnaposto* presenti in una stringa con dati corrispondenti per la creazione di una nuova stringa;

ricerca: operazione mediante la quale si ricerca la presenza, o l'assenza, di un carattere o di una stringa all'interno di un'altra stringa.

Nel paragrafo 5, durante l'esposizione dei problemi da risolvere e delle relative soluzioni, vedremo come i linguaggi esaminati rendono disponibili al programmatore alcune delle operazioni menzionate.

Naturalmente, ai fini del nostro lavoro, sarebbe inutile non poter salvare delle stringhe più o meno complesse in un file, così come lo sarebbe non poter memorizzare il contenuto di un file di testo in una o più stringhe; vedremo pertanto, sempre nel paragrafo 5, alcune istruzioni per salvare dette stringhe in maniera permanente o acquisirle da un file.

4 La tecnologia dei template

Come accennato nel paragrafo precedente, una volta che sappiamo come trattare le stringhe è immediato pensare di scrivere un programma in grado di memorizzare un documento il cui testo sia contenuto in una o più stringhe e in cui eventualmente sostituire le parti variabili con il contenuto di stringhe impostate durante l'esecuzione. Infatti tramite le operazioni sulle stringhe è possibile costruire in modo efficiente brani testuali. Man mano che si utilizzano queste tecniche con successo diviene tuttavia chiaro che nel programma non esiste una separazione concettuale tra dati e testo.

Esaminiamo il caso di un classico problema di *mail merge*, ovvero il problema di produrre una serie di lettere dal contenuto comune da personalizzare a partire dai destinatari e dai relativi indirizzi. Possiamo certamente separare in file distinti l'elenco degli indirizzi, la logica del programma e il testo della lettera ma, usando le tecniche di manipolazione delle stringhe, è proprio il concetto di *modello di lettera* che rimarrà "nascosto" nel codice, così come la sua realizzazione fisica, cioè delle variabili di tipo stringa contenenti il testo da trattare da parte del programma. Ciò rende difficoltoso apportare qualunque modifica al modello limitando l'evoluzione dei contenuti e il riuso del codice.

4.1 Cos'è un template

Il *template*, o *modello*, è un particolare tipo di documento testuale in cui una parte fissa e immutabile di contenuto si alterna a parti variabili chiamate *tag* che contengono nomi individuati da sequenze di caratteri detti delimitatori.

Il template replica nel campo informatico il concetto di *modulo da compilare* con l'ovvia differenza che gli spazi bianchi da riempire sono entità associate al contenuto prodotto da un'elaborazione. A differenza di quanto detto sulle stringhe nel paragrafo precedente, il metodo di lavoro associato ai template è differente perché:

- il template introduce il modello del testo, un nuovo concetto per rappresentare il testo finale;
- il testo finale è la composizione delle parti letterali di testo del template e l'espansione dei tag con informazioni esterne — dette *contesto* — corrispondenti per struttura e nomi;

- un template può essere annidato in altri template per rappresentare un documento strutturato;
- il codice del programma è incentrato sulla logica e diviene più semplice;
- la generazione di testo con il template può essere molto efficiente in funzione dell'implementazione che ne viene data in un linguaggio di programmazione.

La corrispondenza tra template e contesto deve essere biunivoca in termini sia di struttura che di nomi, costituendo l'*interfaccia* tra i dati e la forma che assumeranno nel documento finale.

4.2 I tag

All'interno del template i tag sono introdotti tramite i caratteri prestabiliti dei *delimitatori*. Di solito il delimitatore di apertura è una coppia di parentesi graffe aperte `{{` mentre quello di chiusura è una coppia di parentesi graffe chiuse `}}`.

Un esempio di template minimale è la seguente stringa dove un tag è associato a una variabile del contesto chiamata `name`:

```
"Hello {{ name }}!"
```

I delimitatori — che possono essere reimpostati dall'utente — e gli spazi presenti all'interno dei tag non verranno inseriti nel testo finale. Si tratta infatti di una sorta di ambiente codice. Come vedremo, i tag possono essere di diverso tipo.

4.3 Utilizzo dei template

In termini generali l'uso dei template comprende due fasi successive:

compilazione un processo d'elaborazione del template che fornirà un nuovo componente *binario*;

resa (*rendering*) una o più esecuzioni del componente binario con dati d'ingresso strutturati e corrispondenti che forniranno altrettanti testi finali.

In alcuni casi, la fase iniziale d'elaborazione del template viene svolta durante la compilazione del programma, anziché a ogni sua esecuzione, con evidente incremento dell'efficienza: il programma dovrà eseguire solamente la produzione dei documenti finali. Nei casi tipici dei sistemi basati sui linguaggi di scripting, invece, il template può essere compilato ogni volta che viene unito ai dati. È da tener presente inoltre che, per la produzione da programma di file sorgenti per il sistema TEX, non è tanto cruciale l'efficienza di elaborazione quanto la semplicità e l'espressività della sintassi dei tag del template: l'utente può modificare più agevolmente i documenti generati sia durante la

fase di costruzione che in quella di affinamento o al mutare delle esigenze o delle specifiche.

Occorre un'apposita libreria per utilizzare i template con un dato linguaggio senza eccessivi sforzi programmatici. Sorprendentemente ci sono molte librerie poiché questa tecnologia è molto diffusa sui server di rete per fornire all'utente pagine web dinamiche o file di puro testo scaricabili.

Nelle prossime sezioni verranno illustrati esempi d'uso dei template sia per Lua con la libreria **lustache** (OLIVINE-LABS, 2015), sia per Python con la libreria **Jinja** (Jinja). Questi due *template engine* sono semplici da usare essendo scritti per un linguaggio di scripting e sono adatti alla generazione di sorgenti TEX perché consentono di cambiare i delimitatori di default a doppie parentesi graffe con altri che non interferiscono con il sorgente finale.

5 Applicazioni pratiche

I prossimi paragrafi raccolgono tutta la parte pratica dell'articolo. Ogni paragrafo verterà su uno specifico linguaggio, traducendo in pratica gli elementi teorici visti nei due precedenti paragrafi e mostrando degli esempi concreti in cui un programma genera file di testo contenente codice che uno dei motori di composizione del sistema TEX tradurrà in documenti finiti.

Al termine degli esempi didattici in Lua e Python, due ulteriori paragrafi riporteranno due applicazioni reali. Ci proponiamo con esse di dare indicazioni utili all'utente per definire il progetto nella sua fase iniziale e a svilupparlo successivamente per adattarsi ai cambiamenti nelle sorgenti dati causati da nuove o inaspettate esigenze di gestione o documentazione.

La qualità del codice molto spesso dipenderà anche dall'equilibrio tra le tre componenti principali del processo di generazione: il programma, il template e il sorgente TEX. Per esempio, non è sempre chiaro se far calcolare le somme economiche dal programma o dal template, oppure se introdurre delle macro LATEX per svolgere compiti che potrebbero essere affidati al template. È buona norma basarsi sull'esperienza lasciandosi ispirare dal criterio di buona manutenibilità e sviluppo del progetto di documentazione e da quello dell'efficienza di esecuzione.

Per la comprensione del codice d'esempio e delle applicazioni presentate nell'articolo è necessario conoscere almeno i fondamenti dei linguaggi di programmazione che chiaramente non possono essere forniti qui. Per Lua si consiglia la lettura del libro IERUSALIMSKY (2013) e del manuale di riferimento pubblicamente consultabile sul sito ufficiale. Per Python esistono molti libri e tutorial alcuni dei quali liberamente disponibili in rete. Per esempio, *Think Python* (DOWNEY, 2012), con particolare attenzione riguardo alla versione 3 del linguaggio, tra

l'altro composto in L^AT_EX. Oppure ancora si può consultare il sito <http://getpython3.com/> dove è possibile documentarsi sulle basi del linguaggio — per esempio con il libro scaricabile *Dive into Python 3* (PILGRIM, 2009) — e su temi specifici come la gestione Unicode delle stringhe. Il riferimento per il C è senz'altro il Kernighan & Ritchie (KERNIGHAN e RITCHIE, 2004) ma vale la pena consultare anche il KLEMENS (2014) per tenersi aggiornati sulle modernizzazioni del linguaggio, specialmente sugli ampliamenti ai tipi di dato e all'elaborazione delle stringhe. Infine, per bash è possibile consultare i siti ABS guide, GNU Bash e il manuale in linea (`man bash` da un qualunque terminale `*nix`).

6 Bash

Bash è un interprete di linguaggio sh-compatibile¹ che esegue comandi letti dallo standard input o da un file. Bash incorpora anche delle caratteristiche utili mutuare da Korn e C shell (ksh e csh).

Descriveremo alcune delle operazioni realizzate da uno script bash durante la compilazione di un numero di ArsTeXnica. Avremo modo di osservare che tutte le operazioni si basano sulla creazione di una stringa di testo che poi verrà inserita in una posizione specifica di un file `.tex` predisposto. La manipolazione delle stringhe² in bash è senz'altro più agevole della manipolazione dei numeri, però avremo spesso bisogno di ricorrere a programmi di sistema³ per isolare quelle parti di una stringa di nostro interesse.

Il sorgente che segue, una porzione minima dello script di compilazione di ArsTeXnica, determina l'anno della rivista, genera l'ISSN adeguato alla versione (a stampa e online) e, oltre alla generazione del codice a barre, inserisce i dati nella seconda di copertina.

```

1  #Determina l'anno corrente; serve per
2  #l'add-on dell'issn e, poi, per i dati
3  #di uscita di ArsTeXnica
4  current_year=`date +%Y`
5  ylf=`echo $current_year | cut -b 4`
6
7  #Estrae i dati dell'uscita corrente
8  #e determina l'add-on
9  current_issue=`pwd | awk -F / '{print $NF}'`
10 issue_name="arstexnica_"$current_issue
11 issue_number=`echo $current_issue | \
12   cut -d_ -f2`
13 if [[ $issue_number -lt 10 ]]; then
14   add_on=$ylf"000"$issue_number;
```

1. Mentre sh sta per Bourne shell, bash sta per Bourne-again shell.

2. Di fatto, le uniche operazioni immediate di manipolazione di stringhe in bash sono la concatenazione e il confronto.

3. La locuzione *programmi di sistema* indica quei filtri — semplici programmi a riga di comando che compiono un'operazione piuttosto specifica — normalmente presenti nei sistemi Unix sotto le directory `/bin`, `/usr/bin`, `/usr/local/bin`.

```

15 elif [[ $issue_number -lt 100 ]]; then
16   add_on=$ylf"00"$issue_number;
17 elif [[ $issue_number -lt 1000 ]]; then
18   add_on=$ylf"0"$issue_number;
19 else add_on=$ylf$issue_number;
20 fi
21
22 #Analizza la modalità di generazione,
23 #crea i file appropriati e genera
24 #l'issn corretto
25 lv=`cat latest_version`
26 if [ "$1" = "print" ]; then
27   issn="977182823500 $add_on"
28   echo "Genero issn per $issn..."
29   cat seconda_head.tex issn_print.tex \
30     seconda_tail.tex > seconda.tex;
31   barcode -E -e ean -u cm -g 4x1.5 -b \
32     "$issn" -o issn_code.eps
33 if [ "$lv" = "online" ]; then
34   sed -i arstexnica.tex -e \
35     's/%colorlinks/colorlinks/g; \
36     s/%citecolor/citecolor/g; \
37     s/%linkcolor/linkcolor/g; \
38     s/%urlcolor/urlcolor/g'
39 fi
40 elif [ "$1" = "online" ]; then
41   issn="977182823600 $add_on"
42   echo "Genero issn per $issn..."
43   cat seconda_head.tex issn_online.tex \
44     seconda_tail.tex > seconda.tex;
45   barcode -E -e ean -u cm -g 4x1.5 -b \
46     "$issn" -o issn_code.eps
47 if [ "$lv" = "print" ]; then
48   sed -i arstexnica.tex -e \
49     's/colorlinks/%colorlinks/g; \
50     s/citecolor/%citecolor/g; \
51     s/linkcolor/%linkcolor/g; \
52     s/urlcolor/%urlcolor/g'
53 fi
54 fi
55 sed -i arstexnica.tex -e "s/XX, YY/ \
56   $issue_number, $current_year/g"
```

Vediamo in dettaglio cosa fa questo codice ricordando che le righe inizianti con `#` sono dei commenti. La riga 4 fa sì che venga eseguito per primo il comando (esterno) `tra apici inversi` e questo estrae l'anno (di quattro cifre) dalla data corrente. La stringa così ottenuta (non è un numero) viene assegnata a una variabile. La riga 5 assegna a una seconda variabile l'ultima cifra dell'anno. Per ottenere questo risultato in bash occorre inviare l'output del comando esterno `echo` (che stampa a video una riga di testo) sulla variabile appena assegnata, cioè proprio la stringa-anno, al comando esterno `cut`, che rimuove delle sezioni da ogni riga di un file stampando le rimanenti; in questo caso, stampa il solo quarto byte dopo aver tolto gli altri. La semantica della riga di codice appena vista è: esegui per primo il comando entro gli apici retroversi e assegna il suo output alla variabile.

La riga 9 assegna a una variabile il nome della directory corrente, quella contenente l'u-

scita. Il nome convenzionale di tale directory è `Numero_<numero>`. Dunque la variabile, ipotizzando di parlare del numero 18, conterrà la stringa `Numero_18`. La riga 10 crea una variabile (sempre stringa) a cui assegna il nome dell'uscita così composto: `arstexnica_<contenuto della variabile assegnata alla riga 9>`. Nelle righe 11–12 estraiamo il numero dell'uscita e lo assegniamo a una variabile (c'è da ripetere “di tipo stringa”?) col solito meccanismo `echo + cut`. Ormai avremo notato che l'assegnazione di una variabile in `bash` implica la scrittura del solo nome della variabile; per recuperare il valore in essa contenuta dobbiamo anteporre il simbolo dollaro (\$) al nome della variabile.

Ora abbiamo tutti i dati per formare quel dato variabile del codice ISSN del giornale noto come *add-on*: ultima cifra dell'anno e numero. Questi due numeri dovranno essere separati dal numero di zeri necessario per formare un numero di cinque cifre. Dunque le righe 13–20 si incaricano di confrontare (mediante conversione stringa-numero implicita nelle doppie parentesi quadre dell'`if`) il numero di uscita con 10, 100 e 1000. Per un numero, essere minore di una di quelle cifre, implica essere composto da una, due o tre cifre. Sfruttiamo questo dato per iniettare nella variabile *add-on* il numero opportuno di zeri.

Siamo pronti a iniziare la generazione del numero. Possiamo compilare la rivista in due modalità: stampa (`print`) o elettronica (`online`). Inseriamo in una variabile la modalità dell'ultima compilazione effettuata (salvata in un file di testo, riga 25). Stabiliamo la modalità di compilazione corrente analizzando il parametro fornito al comando (righe 26 e 40) e stabiliamo di conseguenza l'ISSN corretto (righe 27 e 41; l'*add-on* è lo stesso per entrambe le versioni). Quindi generiamo il file della seconda di copertina, dato dalla concatenazione di tre file di testo tramite il comando esterno `cat` (righe 29–30 e 43–44), e il codice a barre (comando esterno `barcode`, righe 31–32 e 45–46).

A seconda che la compilazione corrente sia per la versione a stampa o elettronica e l'ultima eseguita sia stata l'opposta (righe 33 e 47), bisogna commentare o decommentare le impostazioni relative ai link colorati nel file `arstexnica.tex`: le vogliamo nella versione elettronica ma non in quella a stampa (righe 34–38 e 48–52). Naturalmente, se la compilazione corrente ha la stessa modalità della precedente, non c'è bisogno di fare alcunché dato che le impostazioni sono già quelle necessarie. Anche quest'operazione è affidata a un comando esterno: `sed`, o Stream E^Ditor, un editor non interattivo.

L'ultima operazione è inserire i dati relativi al numero e all'anno della rivista nel master `arstexnica.tex`. Di nuovo grazie a `sed` sostituiamo i dati contenuti nelle due variabili costruite con

`bash` alle due stringhe di comodo `XX` e `YY` contenute nel master `.tex` (righe 55–56), in questo frangente strutturato come un modello i cui campi variabili non sono delimitati da tag ma costituiti da testo di comodo.

C'è da chiedersi con quale difficoltà e in quanto tempo avremmo potuto realizzare questo pezzetto di script se avessimo dovuto elaborare dei file binari?

7 C

In questo paragrafo vedremo come istruire un programma C a scrivere parte di un documento L^AT_EX. Per questo esempio didattico scegliamo di calcolare i primi n numeri di Fibonacci e di scriverli in una tabella immagazzinata in un file. Per dare un po' più di informazioni ai lettori, forniremo un altro esempio, sempre didattico, in cui generiamo casualmente un diagramma *scatter* (a dispersione) di cui scriveremo il grafico in un template TikZ (TANTAU, 2015).

7.1 C e tabelle L^AT_EX

Il primo esempio mostrato calcola i primi n numeri di Fibonacci, che ricordiamo essere dati dalla seguente formula (ricorsiva perché calcolata in termini di sé stessa):

$$f(n) = \begin{cases} n & \text{se } n \leq 1 \\ f(n-1) + f(n-2) & \text{altrimenti} \end{cases}$$

La più banale funzione C che calcoli questo algoritmo è la seguente:

```

1 long
2 fibonacci (long numero)
3 {
4     if (numero < 0)
5     {
6         printf ("Numero errato\n");
7         exit (0);
8     }
9     if (numero <= 1)
10        return numero;
11    else
12        return (fibonacci (numero - 1) +
13                fibonacci (numero - 2));
14 }
15 }
```

Vediamo la perfetta corrispondenza dei `return`, cioè della funzione che restituisce il valore d'uscita di una funzione (o del programma se la funzione è `main()`) con le definizioni della formula.⁴ Abbiamo scritto una funzione C ricorsiva. Dal momento che le funzioni ricorsive sono molto onerose dal punto

4. Stiamo volutamente tralasciando una banale questione di efficienza: scambiando le due condizioni dell'`if` rendiamo il programma leggermente più efficiente perché saranno più numerosi i casi in cui calcoliamo la funzione di numeri superiori a 1, dunque non saltiamo una parte di codice per andare al ramo `else`.

di vista dei tempi di calcolo e dell'impiego di CPU, decidiamo di usare una versione iterativa della funzione appena vista:

```

1  long
2  fibonacci (long numero)
3  {
4      if (numero < 0)
5          {
6              printf ("Numero errato\n");
7              exit (0);
8          }
9      if (numero <= 1)
10         return numero;
11     else
12     {
13         int menouno = 1, menodue = 0, i;
14         long res = 0;
15         for (i = 2; i <= numero; i++)
16             {
17                 res = (menouno + menodue);
18                 menodue = menouno;
19                 menouno = res;
20             }
21         return res;
22     }
23 }
```

Non stiamo qui a discutere la correttezza⁵ o l'efficienza⁶ di questa versione della funzione, non essendo questo il luogo deputato. La useremo però per costruirci intorno un programma⁷ che prenda dalla riga di comando il numero di numeri di Fibonacci da calcolare e il nome del file in cui salvare il codice L^AT_EX e termini senza output ma abbia creato una tabella perfettamente inseribile in un documento L^AT_EX e contenente i dati calcolati.

Vediamo il programma principale (di fatto, la funzione main() e poco altro):

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  long fibonacci (long);
6  FILE *open_file (char *, char *);
7
8  int
9  main (argc, argv)
10     int argc;
11     char *argv[];
```

5. Pur non spiegando i dettagli della funzione iterativa, i risultati delle due versioni sono identici.

6. La velocità di calcolo sembra un argomento capzioso in un banale esempio didattico, eppure la versione iterativa del programma finale, scrittura del file compresa, termina l'esecuzione del calcolo dei primi 45 numeri di Fibonacci in un millisecondo sulla macchina di riferimento, quella ricorsiva in circa 21 *secondi* (circa 21 000 volte più lenta).

7. Il programma, didattico, non tiene conto di molti aspetti cruciali in un programma di produzione quali, per esempio, il controllo preventivo sull'input, il controllo degli *overflow*, l'interfaccia utente e il controllo sull'esistenza e l'eventuale volontà di sovrascrivere i file. Quello che conta, ai fini dell'articolo, è mostrare come le funzioni di libreria del C permettono di gestire le stringhe e di scriverle su file.

```

{
    long i, numero;
    char ft[50];
    FILE *filetab;

    numero = atoi (++argv);
    strcpy (ft, ++argv);
    strcat (ft, "_tabella.tex");
    if ((filetab =
        open_file (ft, "w")) == NULL)
        exit (1);
    fprintf (filetab,
        "\\begin{tabular}{rr}\\n\\hline\\n");
    fprintf (filetab,
        "\\textsc{i} & \\textsc{i-esimo} \\
        numero di Fibonacci}\\n\\n");
    fprintf (filetab, "\\hline\\n");
    printf
        ("Calcola i primi %d numeri di \\
        Fibonacci\\n", numero);
    for (i = 0; i < numero; i++)
        fprintf (filetab, "%ld & %ld\\n\\n",
            i + 1, fibonacci (i));
    /*printf("Scrittura del file %s\\n", ft); */
    fprintf (filetab, "\\hline\\n");
    fprintf (filetab, "\\end{tabular}\\n");
    fclose (filetab);
    return 0;
}

long
fibonacci (long numero)
{
    if (numero < 0)
    {
        printf ("Numero errato\n");
        exit (0);
    }
    if (numero <= 1)
        return numero;
    else
    {
        int menouno = 1, menodue = 0, i;
        long res = 0;

        for (i = 2; i <= numero; i++)
        {
            res = (menouno + menodue);
            menodue = menouno;
            menouno = res;
        }
        return res;
    }
}

FILE *
open_file (char *nomefile,
            char *accesso)
{
    FILE *fp;

    if ((fp =
        fopen (nomefile,
            accesso)) == NULL)
        fprintf (stderr,
```

```

77         "Errore nell'apertura di %s\n",
78         nomefile);
79     return fp;
80 }

```

e discutiamone il funzionamento, con particolare riferimento alle istruzioni di manipolazione delle stringhe e gestione dei file.

Le prime tre righe includono delle librerie necessarie perché forniscano dei comandi inesistenti nel C.⁸ La quinta e la sesta riga dichiarano due funzioni: `fibonacci()`, che prende un input di tipo `long int` (intero di quattro byte con segno) e restituisce un risultato di tipo `long int` (`int` è implicito e, pertanto, omesso) e `open_file()`, che prende in input due variabili di tipo puntatore a carattere (stringhe) e restituisce un puntatore a file. La definizione⁹ delle funzioni avverrà dopo la funzione `main()`, cioè dopo il programma principale. Alle righe 8–9 troviamo proprio la dichiarazione della funzione `main()` che prende in input degli argomenti (e `argc` e `argv` sono i nomi standard, ma non obbligatori, delle variabili contenenti il numero di argomenti ricevuto sulla riga di comando e le stringhe contenenti gli argomenti effettivi) e restituisce in output un valore intero con segno.¹⁰ Le righe 10 e 11 dichiarano le variabili di input di `main()`, quindi con la successiva parentesi graffa (riga 12) inizia il corpo del programma. Nelle righe 13–15 vengono dichiarate alcune variabili: due `long int`, una stringa (il C considera gli array di caratteri come stringhe. Notate che la lunghezza massima è predeterminata — il C lo prevede esplicitamente, a meno di allocare dinamicamente lo spazio durante l'esecuzione — e, cattiva pratica, è scritta numericamente nella variabile. Il prossimo esempio mostrerà una pratica migliore) e un puntatore a file.

Come abbiamo detto, i parametri d'ingresso sono visti come delle stringhe; poiché il primo parametro è un numero, bisogna effettuare una conversione. Dunque la riga 17 esegue la conversione: `atoi()` (*array [of char] to integer*) prende in input una stringa e restituisce un numero, corrispondente al numero rappresentato nella stringa. In realtà la

8. Il C, linguaggio molto efficiente perché creato per scrivere sistemi operativi, è anche molto avaro di comandi. Moltissimi comandi usati quotidianamente sono forniti dalla libreria standard, parte integrante del linguaggio. Molti programmatori in COBOL e altri linguaggi “storici” polemizzano proprio su questa “avarizia” di comandi, quando i “loro” standard forniscono decine di comandi e funzioni per molte operazioni fondamentali quali l'I/O.

9. La differenza tra dichiarazione e definizione è, in sostanza, che la prima serve a indicare una forma (nome della variabile o della funzione e tipi di ingresso e uscita) mentre la seconda un contenuto (valore della variabile o serie di istruzioni della funzione e nomi delle variabili di input e output).

10. A differenza di `long int`, che è definito come un intero con segno di 4 byte, `int` è definito come un intero con segno di 2 o 4 byte. La dimensione dipende dall'implementazione.

sintassi dell'istruzione è più complessa: dice, letteralmente, “poiché `argv` punta all'inizio del primo elemento sulla riga (cioè il nome del programma in esecuzione), spostati al secondo elemento (il ++ è un incremento di 1; se postfisso, viene eseguito dopo che la funzione ha preso in input, e processato, il parametro; se prefisso, prima viene eseguito l'incremento e poi passato il parametro alla funzione e processato. Nel caso in esame, incrementare di 1 significa saltare al parametro successivo) e convertilo a numero”.

La riga 18 copia il terzo elemento della riga di comando (secondo parametro, nome del file L^AT_EX contenente la tabella finale) nella variabile `ft` (`strcpy` sta per *string copy*). Quindi la riga 19 completa il contenuto della variabile `ft`, cioè concatena al suo contenuto (`strcat` sta per *string concatenate*) la stringa `_tabella.ft` (per una discussione più approfondita sulla tecnica della concatenazione delle stringhe, si veda più avanti il paragrafo 8.4). Naturalmente, se il nome finale eccede i 49 caratteri (il cinquantesimo serve a contenere il \0 di fine stringa), verrà tagliato. Le righe 19–21, aprendo in scrittura il file che conterrà la tabella L^AT_EX, dicono letteralmente: “se il puntatore a file assegnato alla variabile `filetab` dalla funzione `open_file()` (la discuteremo alla fine) è NULL (cioè l'apertura del file in scrittura non è andata a buon fine) termina l'esecuzione (riga 22) e esci dando in output il valore 1”. Ottenere un valore diverso da NULL significa che l'apertura è andata a buon fine e possiamo andare avanti con l'esecuzione.

Le righe 23–28 (una è interrotta con un \ per motivi di impaginazione; è il modo in cui il C ci permette di mandare a capo delle sequenze di caratteri tra doppi apici, cioè delle stringhe, senza dare errore) scrivono nel file le prime righe del codice L^AT_EX della tabella. Essendo bufferizzata, la scrittura potrebbe non avvenire nel momento esatto in cui l'istruzione viene eseguita. Le righe 29–31 sono solo un *memento* per l'utente: stampano a video che il programma calcola i primi *n* numeri di Fibonacci. Tale stampa è sempre bufferizzata. Le righe 32–34 calcolano effettivamente i numeri voluti e contemporaneamente li scrivono nella tabella (nella riga 33 vediamo l'uso della tecnica dei campi variabili, discussa in maggior dettaglio più avanti, al paragrafo 8.3). Ovviamente `fibonacci(0)` rappresenta il primo numero di Fibonacci, ecco il perché dell'*i+1* nella `fprintf()`. La riga 35, sempre un *memento* per l'utente, è commentata e pertanto non eseguita.

Le righe 36 e 37 scrivono nel file tabella i comandi di chiusura della stessa; la riga 38 chiude il file e la riga 39 termina l'esecuzione del programma restituendo il valore 0 che, per convenzione, in C indica il buon fine.

Vediamo ora la funzione `open_file()`. Poiché la funzione deputata del C (`fopen()`) non segnala

TABELLA 1: Risultato del calcolo dei primi n numeri di Fibonacci.

I	I-ESIMO NUMERO DI FIBONACCI
1	0
2	1
3	1
4	2
5	3
6	5
7	8
8	13
9	21
10	34
11	55
12	89
13	144
14	233
15	377
16	610
17	987
18	1597
19	2584
20	4181

niente all'utente ma solo al programma tramite il parametro di ritorno, la nuova funzione colma questa lacuna stampando a video l'eventuale errore in apertura. Notiamo che, anziché una `printf()`, abbiamo usato una `fprintf()` (stampa su file) che stampi su `stderr`. Quest'ultimo è uno dei "file" standard (gli altri due sono `stdin` e `stdout`); ridirige sul terminale quello che riceve e lo fa in maniera non bufferizzata, dunque la scrittura a video avviene nel momento esatto in cui l'istruzione viene eseguita. Quindi esce restituendo il puntatore a file (o `NULL` in caso di errore).

Eseguiamo il programma, dopo averlo compilato (per compilarlo in Linux basta `gcc -o nfibonacci nfibonacci.c`), per fargli calcolare i primi 20 numeri di Fibonacci. Il risultato è visibile nella tabella 1.

7.2 C e template di grafici

Vediamo ora un secondo esempio in cui generiamo casualmente una serie di coordinate cartesiane intere comprese tra (0,0) e (10,10) e le riportiamo su un grafico TikZ la cui "ossatura" è contenuta in un modello chiamato `template.tex`. Poiché l'esempio è piuttosto limitato, decidiamo di abbandonare i tag in favore di una riga di commento e di uno spazio da riempire in maniera standard. Di seguito il contenuto del template.¹¹

11. Lasciamo volutamente debordare il codice per far capire al lettore che ogni riga letta dal programma C è esattamente quella qui stampata.

```

1 \begin{tikzpicture}
2 \draw[very thin,color=gray] (0,0.01) grid (10,10);
3 \draw[->] (-0.2,0) -- (10.5,0) node[right] {$x$};
4 \draw[->] (0,-0.2) -- (0,10.5) node[above] {$y$};
5 %---riga da riempire
6 \draw[color=black!30!green] () node {$\times$};
7 \end{tikzpicture}

```

È facile vedere che la seconda riga disegna la griglia del piano cartesiano e le righe 3 e 4 disegnano gli assi del piano. La riga 5 contiene il commento che controlleremo per capire qual è il punto in cui inserire i dati.¹² La riga successiva al commento è quella da riempire coi dati generati e ripetere tante volte quante sono le coordinate generate.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <time.h>
5
6 #define MAXRIGA 2048
7 #define CONTROLLO "%---riga da riempire"
8
9 FILE *open_file (char *, char *);
10
11 int
12 main (argc, argv)
13     int argc;
14     char *argv[];
15 {
16     long i, numero, cr;
17     int x, y;
18     char nomefile[MAXRIGA], fg[MAXRIGA],
19         tpl[] =
20         "template.tex", riga[MAXRIGA];
21     FILE *filegraf, *template;
22
23     numero = atoi (++argv);
24     strcpy (fg, ++argv);
25     strcat (fg, "_grafico.tex");
26     if ((filegraf =
27         open_file (fg, "w")) == NULL)
28         exit (1);
29     if ((template =
30         open_file (tpl, "r")) == NULL)
31         exit (1);
32     srand (time (NULL));
33     while (fgets
34         (riga, MAXRIGA - 1,
35         template) != NULL)
36         if (strncmp (riga, CONTROLLO,
37             strlen(CONTROLLO)))
38             fputs (riga, filegraf);
39     else
40     {
41         fgets (riga, MAXRIGA - 1,
42             template);
43         printf

```

12. Vista la dimensione e la semplicità del template, sarebbe stato più immediato andare direttamente al punto da modificare tramite una `fseek()`; non è però questo quello che vogliamo mostrare, ma un metodo di lavoro più generale che discuteremo più avanti.

```

44         ("Genera %d coppie di coordinate \
45         di un diagramma a dispersione\n",
46         numero);
47     for (i = 0; i < numero; i++)
48     {
49         for (cr = 0;
50             riga[cr] != '('; cr++)
51             fputc (riga[cr],
52                   filegraf);
53         x = (int) rand () % 11;
54         y = (int) rand () % 11;
55         fprintf (filegraf, "(%d,%d",
56                 x, y);
57         for (cr++; riga[cr] != '\n';
58             cr++)
59             fputc (riga[cr],
60                   filegraf);
61         fputc ('\n', filegraf);
62     }
63 }
64 printf ("Scrittura del file %s\n",
65         fg);
66 fclose (template);
67 fclose (filegraf);
68 return 0;
69 }
70
71 FILE *
72 open_file (char *nomefile,
73            char *accesso)
74 {
75     FILE *fp;
76
77     if ((fp =
78         fopen (nomefile,
79               accesso)) == NULL)
80         fprintf (stderr,
81                 "Errore nell'apertura di %s\n",
82                 nomefile);
83     return fp;
84 }

```

Passiamo alla descrizione di questo esempio saltando quelle parti di codice già discusse all'esempio precedente.

Le righe 6 e 7 contengono la direttiva **#define**, cioè un'istruzione che dice al compilatore: "ogni volta che nel sorgente trovi i termini **MAXRIGA** e **CONTROLLO**, sostituiscili con 2048 e "%---riga da riempire". A differenza dell'esempio precedente, in cui il numero 50 era contenuto nel sorgente e la cui sostituzione comportava di cercarlo in tutto il codice (e non è detto che tutte le occorrenze di 50 abbiano lo stesso significato, cioè può darsi che in alcune parti il 50 debba rimanere tale e non essere sostituito), in questo caso, invece, basta modificarlo in una sola riga contenuta all'inizio del sorgente. Le righe 15–19 contengono delle definizioni di variabili *locali* (cioè visibili solo alla funzione in cui sono definite) utili al programma.

Le righe 23–31 sono equivalenti a quanto già visto nell'esempio di Fibonacci. L'unica differenza è che ora apriamo due file: quello che conterrà

il grafico in sola scrittura e quello contenente il template in sola lettura.

La riga 32 inizializza il generatore di numeri casuali prendendo come seme il valore restituito dalla funzione **time()**. Altri linguaggi forniscono una funzione equivalente senza parametri di input chiamata **randomize**.

Le righe 33–63 sono quelle che fanno tutto. Cerchiamo di raccontarle in maniera discorsiva, come se stessimo raccontando una storia. Prima di raccontare, sarà buona cosa descrivere in termini sommari alcune istruzioni introdotte qui per la prima volta:

while è uno dei cicli del C; esegue il corpo del **while** (cioè tutto quanto contenuto nelle parentesi graffe successive o, in mancanza di parentesi graffe, solo l'istruzione seguente) fin tanto che la condizione del ciclo è *vera*. Al contrario, esce;

fgets() legge una riga da un file (di testo) terminata da **\n** o dalla dimensione fornita come secondo parametro;

strncmp() compara i primi *n* caratteri delle due stringhe fornite come argomento;

strlen() restituisce la lunghezza della stringa fornita come argomento;

fputs() scrive una stringa in un file di testo;

for è un altro ciclo del C; la sua sintassi è piuttosto articolata: all'interno delle parentesi tonde *possono* essere fornite tre espressioni, ognuna delle quali obbligatoriamente separata dal **;**. La prima espressione funge da inizializzazione del ciclo, cioè viene eseguita solo all'entrata nel ciclo; la seconda espressione viene eseguita all'inizio di ogni iterazione successiva; infine, la terza viene eseguita alla fine di ogni iterazione. L'espressione **for (; ;)** è perfettamente lecita in C e viene usata per eseguire cicli infiniti;

rand() restituisce un valore intero, compreso tra 0 e **RAND_MAX**, (pseudo)casuale estratto da una lista di numeri eventualmente generata da **srand()**;

% calcola la "divisione modulo", cioè fornisce il resto di una divisione tra operandi interi.

Ora possiamo raccontare la parte di codice rimanente: per ogni riga letta dal template (33–35) verifichiamo che non sia la riga "di controllo" (36–37) per riscriverla sul file del grafico (38). Nel caso la riga sia quella "di controllo" (39), leggiamo la riga successiva (41–42) e scriviamola nel file del grafico (49–52 e 55–61) tante volte quante sono le coppie di coordinate da generare (47), "iniettando" ogni riga con la coppia di coordinate intere

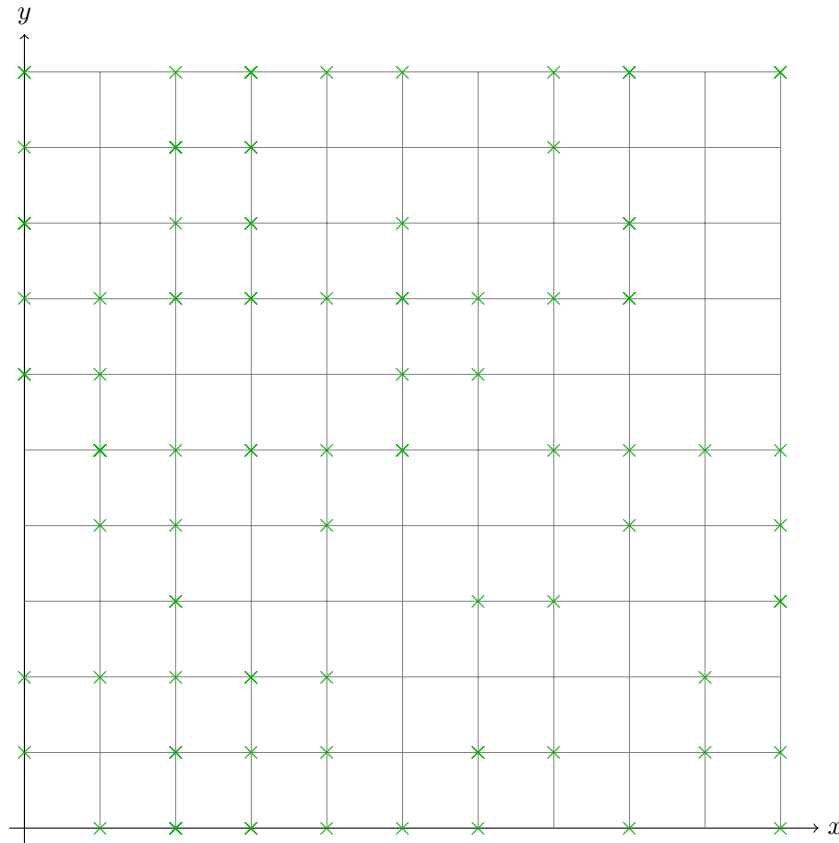


FIGURA 3: Grafico del diagramma scatter relativo alla generazione di 100 coppie di coordinate intere.

generate casualmente e comprese tra (0,0) e (10,10) (53–56) (l’iniezione avviene copiando ogni carattere di input nel file di output quando il carattere non sia ‘(’, quindi scrivendo la stringa “(x,y” per poi copiare i rimanenti caratteri della riga a partire da ‘)’).

Il risultato di un’esecuzione su 100 coppie di coordinate è dato nella figura 3.

8 Lua

Questo paragrafo e il prossimo, i più corposi per informazioni ed esempi di codice forniti, mostreranno diverse tecniche basate sulle stringhe di testo o sulla tecnologia dei template per generare sorgenti TeX.

8.1 Stringhe in Lua

Le stringhe in Lua sono immutabili, caratteristica comune ad altri linguaggi. Questo significa che, una volta creata, una stringa non può più essere modificata. L’unico modo, indiretto, per modificare una stringa è crearne una nuova in sostituzione dell’originale.

Esistono sostanzialmente due modi per introdurre in un programma Lua valori letterali di tipo stringa:

1. delimitando i caratteri con il simbolo di doppio apice " oppure con il simbolo di apice semplice ';

2. usando un delimitatore a doppie parentesi quadre di apertura e chiusura [[e]].

Nel primo caso la stringa occupa un’unica linea mentre eventuali caratteri di controllo come il ritorno a capo `\n` o la tabulazione `\t` verranno convertiti nei rispettivi caratteri ASCII. Nel secondo caso, all’opposto, i caratteri non stampabili non verranno interpretati ma è possibile estendere il testo su più linee. Il codice seguente mostra l’uso di queste due diverse sintassi Lua:

```
local s1 = "This is a \\string"
local s2 = 'Literal "\\\" means "\\"'
local s3 = [[
This is a multiline literal string
And this is a strange sequence: \n\t but
not much...
]]

print(s1)
print(s2)
print(s3)
```

Questo programma produce il seguente output:

```
This is a \string
Literal "\\\" means "\\"
This is a multiline literal string
And this is a strange sequence: \n\t but
not much...
```

Si noti come con il delimitatore a parentesi quadre di Lua venga ignorato l'eventuale primo carattere di ritorno a capo — cosa che succede proprio con la stringa `s3` dell'esempio formata da quattro linee l'ultima delle quali vuota corrispondente al ritorno a capo inserito dopo `not much...`

Infine, volendo inserire nel testo letterale gli stessi delimitatori, nel caso degli apici è possibile usare ancora l'escaping come in `"una \"bella\" parola"`, mentre nel caso dei delimitatori a coppie di parentesi quadre è possibile usare un qualsiasi numero di segni uguali purché bilanciati tra apertura e chiusura come in:

```
[==[una [[bella]] parola]==]
```

8.2 Creare file di testo

In Lua è molto semplice creare un file di testo ricorrendo alla libreria standard del linguaggio e in particolare alle funzioni contenute nel modulo `io`. Il metodo `io.open()` può creare oggetti file specificando come primo parametro il nome del file stesso e come secondo parametro una delle seguenti modalità: lettura `r`, scrittura `w` o scrittura con aggiunta `a`.

Se il parametro di apertura è la stringa `"w"` il file verrà aperto con i diritti di scrittura e, se già esistente, verrà cancellato. Ecco un esempio di codice completo il cui risultato di esecuzione è quello di creare il più piccolo sorgente T $\TeX$ possibile:

```
-- the most simple source file in TeX
local filename = "simple.tex"
local content = "Hello world!\n\\bye"

-- a new file in the current directory
-- opened in 'write mode':
local file = assert(io.open(filename, "w"))
file:write(content)
file:close()
```

Il contenuto del file sarà esattamente il seguente:

```
Hello world!
\\bye
```

Per creare un testo contenente dati variabili occorre necessariamente elaborare stringhe utilizzando funzioni di libreria o la concatenazione. Dedicheremo le prossime due sezioni a ciascuno di questi due metodi.

8.3 Campi variabili

La funzione della libreria standard di Lua che produce stringhe con campi variabili è `string.format()`. Il primo argomento della funzione è una "stringa formato" e gli argomenti successivi sono i valori corrispondenti ai campi. Questo breve esempio stampa una riga di codice L $\TeX$ in modalità matematica:

```
local fmt = "\\( \\pi = %0.5f \\)"
local s = string.format(fmt, math.pi)
print(s) --> '\\( \\pi = 3.14159 \\)'
```

Possiamo notare che il campo variabile è relativo a un numero in virgola mobile arrotondato alla quinta cifra decimale.

I campi variabili, già visti nel paragrafo relativo al C pur senza averne specificato il nome, sono delle sequenze formate dal simbolo di percentuale e da un carattere che identifica il tipo di dato: tra gli altri, `d` i numeri interi, `s` le stringhe, `f` i numeri in virgola mobile. Come abbiamo visto è possibile inserire tra questi due elementi ulteriori specificatori per l'arrotondamento numerico o il riempimento di campi a lunghezza fissa. Tutte queste particolarità — naturalmente documentate nel manuale di riferimento del linguaggio — derivano da funzionalità già presenti nel linguaggio C e quindi immediatamente comprensibili per un buon numero di sviluppatori.

8.4 Concatenazione di stringhe

La concatenazione di stringhe comporta la creazione di una nuova stringa pertanto, per motivi di efficienza e d'utilizzo di memoria, è bene ricorrervi per la concatenazione di un numero piccolo di stringhe al di fuori di cicli iterativi. L'operatore di unione è rappresentato da due caratteri punto consecutivi:

```
local s = "Hello" .. " " .. "world!"
assert( s == "Hello world!")
```

La costruzione di un sorgente L $\TeX$ richiede l'unione di molte stringhe perciò è molto vantaggioso introdurre un'altra funzione della libreria standard chiamata `table.concat()`. Con essa possiamo concatenare un numero anche molto grande di stringhe, per esempio per formare una lunga lista di dati.

Il codice di esempio completo, dove una serie numerica di venti numeri casuali viene assemblata in un'unica stringa, è il seguente:

```
local data = {}
for i=1,20 do
    data[#data + 1] = string.format(
        "%d = %0.3f",
        i,
        math.random()
    )
end

local content = table.concat(data, "\n")
print(content)
```

8.5 Considerazioni finali sulle stringhe

Basandoci sulla corrispondenza tra tipo stringa e file di testo abbiamo presentato le tecniche fondamentali per la costruzione di un sorgente. Queste tecniche sono due: la sostituzione di campi formato all'interno delle stringhe costituiti da *chiavi* segnaposto e la concatenazione efficiente di stringhe per mezzo di un buffer.

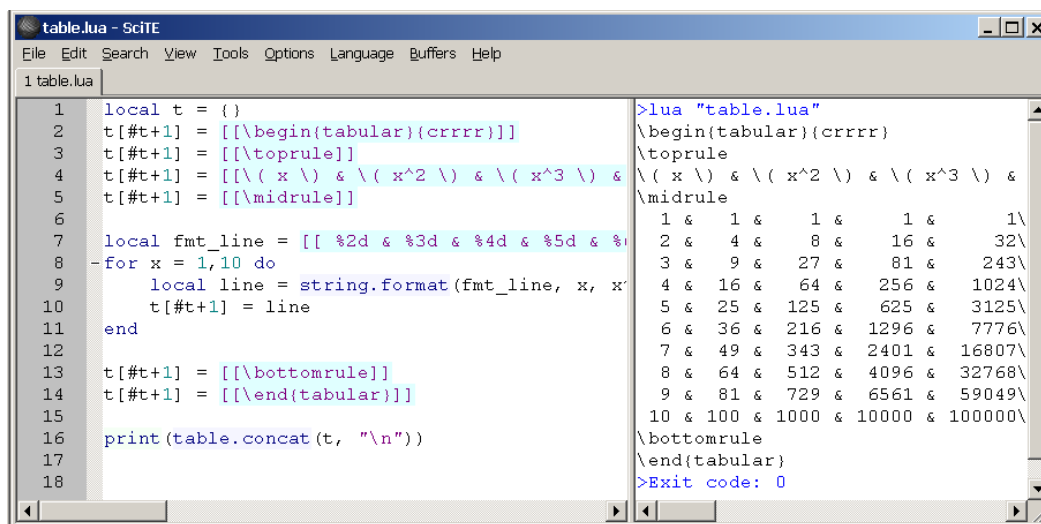


FIGURA 4: L'editor SciTE in azione mostra nel pannello laterale l'uscita del programma Lua per generare il codice $\text{\LaTeX}$ della tabella numerica descritta nel testo. Per eseguire il codice Lua è sufficiente premere il tasto F5 mentre per aprire e chiudere il pannello di output si usa il tasto F8.

Riassumendo è possibile costruire il testo attraverso il tipo stringa — tipo immutabile in molti linguaggi di programmazione — con le seguenti operazioni:

- stringhe letterali;
- campi formato (interpolazione);
- concatenazione efficiente tramite buffering.

Nelle prossime sezioni saranno presentati esempi concreti che utilizzano le operazioni sulle stringhe per generare sorgenti $\text{\LaTeX}$. Si tratta dell'impiego di funzionalità di base dei linguaggi di programmazione o delle loro librerie standard.

8.6 Come eseguire i programmi Lua

L'esecuzione di uno script Lua avviene allo stesso modo della compilazione di un sorgente $\text{\LaTeX}$: da riga di comando oppure da un editor predisposto tramite pressione di una combinazione di tasti.

Se non si vuole installare un interprete Lua si può far ricorso al programma `texlua` compreso in $\text{\TeX}$ Live, mentre come editor è consigliabile l'utilizzo di SciTE (figura 4), in grado di visualizzare in un pannello laterale l'uscita dei programmi lanciati premendo il tasto F5. L'editor SciTE è molto comodo e può essere programmato internamente in Lua.

8.7 Un esempio: costruire una tabella

Ci proponiamo di costruire il codice $\text{\LaTeX}$ corrispondente a una tabella numerica in cui nella prima colonna compare una variabile indipendente e nelle successive i suoi corrispondenti valori delle potenze fino alla quinta.

Ci avvarremo di una tabella Lua dove memorizzare riga per riga tutto il codice $\text{\LaTeX}$ per poi stampare il risultato in console. Il codice mostrato

di seguito, la cui uscita è mostrata nella figura 4, è suddiviso in tre parti:

1. inserimento del codice di apertura dell'ambiente `tabular` e della riga d'intestazione della tabella;
2. inserimento tramite un ciclo iterativo delle righe di contenuto numerico;
3. inserimento del codice di chiusura dell'ambiente `tabular`.

```
-- a new table called 't'
local t = {[[{\begin{tabular}{crrrr}}]}
t[#t+1] = [[{\toprule}]
t[#t+1] = [[{\(x\) & \((x^2\) & \((x^3\) & \((x^4\) & \((x^5\) & \]}
t[#t+1] = [[{\midrule}]

local f = [[ %2d & %3d & %4d & %5d & %6d\]]
for x = 1,10 do
    local line = string.format(
        f, x, x^2, x^3, x^4, x^5
    )
    t[#t+1] = line
end

t[#t+1] = [[{\bottomrule}]
t[#t+1] = [[{\end{tabular}}]]

print(table.concat(t, "\n"))
```

Lo scopo di questo esempio non è presentare una soluzione definitiva — anzi il codice può essere migliorato in molti modi — ma di mostrare il modello d'implementazione basato sul tipo stringa per la produzione automatica di codice: una sequenza d'inserimenti di testo — stringa letterale o dato variabile — in un *buffer*, linea per linea.

8.8 lustache un template engine per Lua

lustache è un'implementazione in Lua del sistema di template *mustache*. Si tratta di specifiche generali consultabili alla pagina (MUSTACHE, 2015) il cui nome fa riferimento alla somiglianza tra le parentesi graffe dei delimitatori e un bel paio di baffi. Tali specifiche sono disponibili in un buon numero di linguaggi grazie a progetti indipendenti. Rimandiamo il lettore al sito della libreria *lustache* (OLIVINE-LABS, 2015) per le istruzioni d'installazione.

Un template per un sorgente T_EX minimo è mostrato nel seguente frammento di codice:

```
local template = [[
Hello {{ my_friend }}!
{{ bye }}]]
```

Per l'esecuzione di questo template occorre fornire al metodo `render()` della libreria *lustache* una tabella con chiavi uguali a `my_friend` e `bye` previste dai due tag. I valori associati a queste chiavi verranno inseriti nel testo finale. Se invece la chiave non è presente, il tag in questione verrà semplicemente ignorato.

Il codice d'esempio completo è il seguente:

```
local lustache = require "lustache"

local template = [[
Hello {{ my_friend }}!
{{ bye }}]]

local data = {
    my_friend = "John Doe",
    bye = "\\bye",
}

local doc = lustache:render(template, data)
print(doc) -- or save it on a .tex file
```

e troveremo in console il testo sperato:

```
Hello John Doe!
\\bye
```

8.9 Una lista

Oltre ai tag semplici ci sono quelli di blocco concettualmente simili agli ambienti di L^AT_EX. Per iterare il testo di una lista, per esempio, si può creare un blocco dopodiché una lista puntata:

```
local lustache = require "lustache"

local tpl = [[
\\begin{itemize}
{{#list}}
    \\item {{ . }};
{{/list}}
\\end{itemize}}]

local book_list = {
    "Guide to LaTeX",
    "The LaTeX Companion",
```

```
    "The LaTeX Graphics Companion",
    "The LaTeX Web Companion",
}

doc = lustache:render(
    tpl,
    {list = book_list}
)
print(doc)
```

Il tag di blocco `list` è composto dal tag di apertura `#list` e da quello di chiusura `/list`. All'interno di questo blocco si trova un tag anonimo, rappresentato dal carattere `'.'`, che riceverà uno alla volta gli elementi contenuti nella tabella/array `list`.

L'uscita del programma, riportata di seguito, mostra alcune imperfezioni:

```
\\begin{itemize}

    \\item Guide to LaTeX;

    \\item The LaTeX Companion;

    \\item The LaTeX graphics Companion;

    \\item The LaTeX Web Companion;

\\end{itemize}
```

8.10 Controllare i ritorni a capo

Eliminare dal template i caratteri di fine riga comporta la loro eliminazione anche dal testo finale; nell'esempio erano stati introdotti per maggiore chiarezza del codice del template. Diversamente possiamo usare i tag commento — tali quando il carattere `'!'` segue i delimitatori di apertura — perché ammettono ritorni a capo al loro interno. Ecco come potremmo riscrivere il template precedente per eliminare le righe bianche di troppo:

```
-- no newline
local tpl = [[
\\begin{itemize}
{{#list}}    \\item {{ . }};
{{/list}}\\end{itemize}]]
```

Forma alternative sono:

```
-- newline with comment tag
local tpl = [[
\\begin{itemize}
{{#list}}{! comment tag
}}    \\item {{ . }};
{{/list}}{!
}}\\end{itemize}]]
```

oppure:

```
-- with tag inner newline
local tpl = [[
\\begin{itemize}
{{#list
}}    \\item {{ . }};
{{/list
}}\\end{itemize}]]
```

8.11 Generalizzare il tipo di lista

Il modo più semplice per generalizzare il tipo di lista è renderla attraverso un tag. Ciò implica modificare i delimitatori a coppie di graffe a causa delle possibili interazioni con le graffe che circondano il nome dell'ambiente, come dimostra il seguente template:

```
local tmpl = [[
\begin{{ env }}
{{#list}} \item {{ . }};
{{/list}}\end{{ env }} ]]
```

Il primo tag verrà interpretato come un tag delimitato a tre graffe che a differenza di quello a due evita l'escaping del contenuto — ne ripareremo più avanti nella sezione 8.15 — dando `\beginitemize` mentre l'ultimo tag darà come ci si aspetta `\end{ itemize }`.

Non è facile trovare dei delimitatori che certamente non compariranno mai in un sorgente `TeX`, che siano compatibili con la sintassi Lua per le stringhe e che rendano il template il più leggibile possibile. In questo esempio abbiamo pensato alla coppia `<| |>`. I delimitatori vengono modificati, direttamente nel template e ogni volta che sarà necessario, tramite un tag e il segno `=`, carattere che non può essere compreso nei delimitatori assieme al carattere spazio.

Nell'esempio di codice completo è possibile notare che i segni di uguale `=` sono esterni ai nuovi delimitatori e che questi devono essere separati almeno da un carattere spazio:

```
local lustache = require "lustache"

local tmpl = [[{{=<| |>=}}<| |>
|>\begin{{<| env |>}}
<|#list|> \item <| . |>;
<|/list|>\end{{<| env |>}}]]

local books = {
  "Guide to LaTeX",
  "The LaTeX Companion",
  "The LaTeX Graphics Companion",
  "The LaTeX Web Companion",
}

doc = lustache:render(tmpl,
  {list = books, env="itemize"})
print(doc)
```

8.12 Rendere speciale l'ultimo elemento

Le specifiche `mustache` non prevedono la gestione speciale dell'ultimo elemento della lista, per esempio per inserire il carattere `'.'` finale al posto del punto e virgola.

Nel caso in cui la chiave prevista dal tag non è presente nei dati, non verrà generato un errore ma semplicemente il testo di sostituzione sarà vuoto. Questo comportamento ci consente di risolvere il problema al prezzo di una modifica dei dati:

```
local lustache = require "lustache"

local tmpl = [[
\begin{itemize}
{{#list}} \item {{ book }}{{!
}}{{^dot}};{{/dot}}{{#dot}}.{{/dot}}
{{/list}}\end{itemize}]]

local list = {
  {book = "Guide to LaTeX"},
  {book = "The LaTeX Companion"},
  {book = "The LaTeX graphics Companion"},
  {book = "The LaTeX Web Companion",
   dot = true},
}

doc = lustache:render(tmpl, {list=list})
print(doc)
```

Nel template sono stati inseriti due nuovi blocchi; il primo sarà eseguito per effetto del carattere `^` se la chiave `dot` non esiste, viceversa il secondo. Così, inserendo solamente nell'ultima tabella il campo `dot`, otteniamo l'effetto di aggiungere il `;` alla fine di tutte le righe tranne l'ultima dove troveremo il carattere punto.

Modificare la struttura dati per ottemperare a difetti del template è un rimedio estremo che molto probabilmente compromette lo sviluppo del progetto proprio perché l'obiettivo primario è quello di separare i dati dal modello del testo.

Per fortuna è possibile avvalerci di alcune funzionalità avanzate di Lua come le *closure* e le funzioni come oggetti di prima classe. Nel codice che segue il problema è risolto senza modificare i dati: una funzione è inserita nel template e sulla base del numero di riga aggiunge il corretto carattere finale:

```
local lustache = require "lustache"

local tmpl = [[
\begin{itemize}
{{#list
}} \item {{ . }}{{ end_char }}
{{/list
}}\end{itemize}]]

local book_list = {
  "Guide to LaTeX",
  "The LaTeX Companion",
  "The LaTeX Graphics Companion",
  "The LaTeX Web Companion",
}

local rows = #book_list
-- closure
function end_char()
  rows = rows - 1
  if rows == 0 then
    return "."
  else
    return ";"
  end
end
```

```
doc = lustache:render(tmpl,
  {list = book_list,
   end_char = end_char,})
print(doc)
```

8.13 Una tabella numerica

Volendo replicare l'esempio della tabella numerica della sezione 8.7 possiamo scrivere il seguente template:

```
local lustache = require "lustache"

local tmpl = [[
\begin{tabular}{crrrrr}
\toprule{! package booktabs required }
\(\ x \) & \(\ x^2 \) & \(\ x^3 \) & \(\ x^4 \) & \(\ x^5 \) \\
\midrule
\{{#rows}
\}\{\{x1\}\} & \{\{x2\}\} & \{\{x3\}\} & \{\{x4\}\} & \{\{x5\}\} \\
\{/rows
\}\}\bottomrule
\end{tabular}]]

local rows = {}
for x = 1,10 do
  rows[#rows+1] = {
    x1 = x,
    x2 = x^2,
    x3 = x^3,
    x4 = x^4,
    x5 = x^5,
  }
end

local data = {rows=rows}
local doc = lustache:render(tmpl, data)
print(doc)
```

Anche in questo caso le caratteristiche avanzate di Lua e della libreria `lustache` possono essere impiegate per offrire una soluzione più elegante variando il dato associato ai nomi `x1`, ... `x5` del template: anziché dati numerici è possibile associare loro delle funzioni.

Questa volta la tabella con i dati per il template, chiamata `data`, conterrà alla chiave `rows` una tabella/array contenente la serie di tabelle con chiave `x`. Tale serie verrà passata in sequenza alle funzioni, e quindi alle chiavi `x1`, ... `x5`, le funzioni di calcolo.

È semplice approfittarne di queste funzioni per aggiungere spazi bianchi in numero sufficiente per incolonnare a destra i valori; a tale proposito utilizzeremo la funzione di libreria `string.format()`.

Il listato seguente implementa quanto detto utilizzando lo stesso template del codice precedente. Il risultato del codice L^AT_EX compilato è riportato nella tabella 2:

```
local lustache = require "lustache"
```

TABELLA 2: La tabella ricavata dal codice Lua riportato nel testo che fa uso della tecnologia dei template implementata nella libreria `lustache`.

x	x^2	x^3	x^4	x^5
1	1	1	1	1
2	4	8	16	32
3	9	27	81	243
4	16	64	256	1024
5	25	125	625	3125
6	36	216	1296	7776
7	49	343	2401	16807
8	64	512	4096	32768
9	81	729	6561	59049
10	100	1000	10000	100000

```
local tmpl = [[
\begin{tabular}{crrrrr}
\toprule{! package booktabs required }
\(\ x \) & \(\ x^2 \) & \(\ x^3 \) & \(\ x^4 \) & \(\ x^5 \) \\
\midrule
\{{#rows}
\}\{\{x1\}\} & \{\{x2\}\} & \{\{x3\}\} & \{\{x4\}\} & \{\{x5\}\} \\
\{/rows
\}\}\bottomrule
\end{tabular}]]

local rows = {}
for i=1,10 do
  rows[#rows+1] = {x=i}
end

local f = string.format
local data = {
  rows = rows,
  x1=function (t) return f("%2d", t.x ) end,
  x2=function (t) return f("%3d", t.x^2) end,
  x3=function (t) return f("%4d", t.x^3) end,
  x4=function (t) return f("%5d", t.x^4) end,
  x5=function (t) return f("%6d", t.x^5) end,
}

local doc = lustache:render(tmpl, data)
print(doc)
```

8.14 Una funzione come tag di un template

La libreria `lustache` ammette che i tag di blocco possano corrispondere a funzioni. Il testo risultante sarà il valore restituito dalla funzione associata al tag.

Cominciamo col mostrare il funzionamento di questa tecnica scrivendo una funzione che restituisce il contenuto del blocco racchiuso in una macro L^AT_EX (il primo argomento tabella può essere ignorato):

```
local lustache = require "lustache"

local tmpl = [[
\{{#bf}}a great idea{\bf}]]
```

```

local doc = lustache:render(
  tmpl,
  {bf = function (tab, rendered)
    return "\\textbf{..rendered..}"
  end}
)
print(doc)

```

Il risultato è il seguente:

```
\textbf{a great idea}
```

Il tag *funzione* può contenere ulteriori tag, per esempio:

```

local lustache = require "lustache"

local tmpl = [[
{{#bf}}a great {{ thing }}{{/bf}}]]

local function enclose_bf(tab, rendered)
  return "\\textbf{..rendered..}"
end

local doc = lustache:render(
  tmpl,
  {thing = "song", bf = enclose_bf}
)
print(doc) --> "\\textbf{a great song}"

```

Questa funzionalità potrebbe essere sfruttata per formattare numeri decimali poiché, nella sua essenzialità, la libreria *lustache* non offre la possibilità di applicare un formato ai dati. L'idea di per sé è corretta perché deve essere il template il componente responsabile dell'aspetto da far assumere ai dati (principio di separazione tra modello e vista).

Nel prossimo esempio di codice questo concetto è implementato con una funzione generale chiamata *format()* perché sia semplice creare ulteriori funzioni di formato come la *num3()*:

```

local lustache = require "lustache"

local tmpl = [[
\\( \pi = {{#num3}}{{pi}}{{/num3}}\\).
\\( e = {{#num3}}2.718281828459{{/num3}}\\).
]]

local function format(x, prec)
  local fmt = string.format(
    "%0.%.df",
    prec
  )
  return string.format(fmt, x)
end

local doc = lustache:render(
  tmpl,
  {pi = math.pi,
   num3 = function (t, x)
     return format(x, 3)
   end}
)

```

```

}
)
print(doc)

```

8.15 Templating ricorsivo

In precedenza abbiamo accennato al fatto che un template può contenerne altri per modellare un testo strutturato. La struttura testuale espressa per mezzo di template ricorsivi è molto potente ma può risultare complessa da gestire.

Nei termini della libreria *lustache* un sub-template viene chiamato *partials*. Esso è un normale template i cui tag devono però far riferimento diretto ai dati forniti al template principale.

Nel template di livello superiore invece, un apposito tag caratterizzato dal segno di maggiore (>) inserito subito dopo il delimitatore di apertura espanderà il template *figlio*. Quest'ultimo dovrà essere contenuto in una tabella passata come terzo argomento alla funzione *render()*.

La costruzione del codice L^AT_EX per inserire una figura come oggetto mobile è un esempio opportuno:

```

\begin{figure}
\centering
\includegraphics[width=\textwidth]{image}
\caption{A great image.}
\label{fig:animage}
\end{figure}

```

Per generare questo testo definiamo un template principale che esprime il modello generale di un ambiente L^AT_EX contenente un secondo template per il contenuto. Modificando al solito i delimitatori di default avremo:

```

local tmpl_env = [[
{{=<| |>=}}\begin{<| env |>}
<|> body |>\end{<| env |>}}]

```

Il template interno accoglierà i due campi variabili opzionali per la posizione centrata dell'immagine e la larghezza imposta e i tre campi obbligatori per il nome del file, il testo della didascalia e l'etichetta di riferimento.

Il fatto che un dato opzionale può non esistere si riflette nel modo in cui il tag associato è inserito nel template, cosa che lo rende meno leggibile, come è possibile constatare nel seguente script (la cosa è specialmente vera per la prima parte del template interno):

```

local lustache = require "lustache"

local tmpl_env = [[
{{=<| |>=}}\begin{<| env |>}
<|> body |>\end{<| env |>}}]

local partials = {body = [[
{{#center}}\centering
{{/center}}\includegraphics{{#width}}
[width={{width}}]{{/width}}{{=<|

```

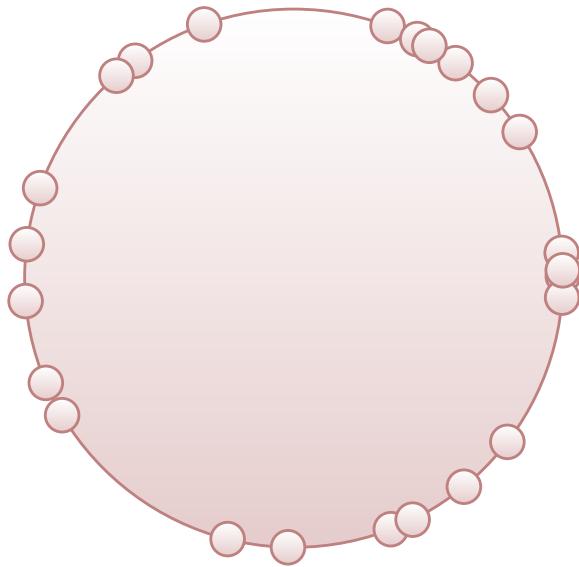


FIGURA 5: Una figura in TikZ il cui codice sorgente è stato generato da un programma che utilizza la tecnologia dei template annidati, scalata rispetto alle dimensioni originali.

```
|>}}{<| file |>}
\caption{<| caption |>}
\label{<| label |>}
]]}

doc = lustache:render(
  tmpl_env,
  {
    env="figure",
    center = true,
    width = "\\textwidth",
    file = "image",
    caption = "A great image.",
    label = "fig:animage",
  },
  partials
)
print(doc)
```

Un'ulteriore discussione sui template annidati è quella che riguarda la realizzazione del codice che produce la figura 5 con l'utilizzo del pacchetto grafico TikZ (TANTAU, 2015). Si tratta di disporre in modo casuale 24 cerchietti su una circonferenza.

La struttura che desideriamo dare al template principale corrisponde a un ambiente con opzioni; il modello realizzato comprende due sottostrutture a cui corrisponderanno altrettanti template interni chiamati rispettivamente `option` e `body`:

```
local tmpl_env = [[
  {<=<% %>=>}}\begin{<% env %><%> option %>
<%> body %>
\end{<% env %>}}]
```

Il primo sub-template, `option`, è semplice: contiene le parentesi quadre che racchiudono un tag associato alla stringa completa il cui codice definisce lo stile TikZ dei cerchietti. Indubbiamente una soluzione che potrà essere sostituita da un'altra più sofisticata.

Poiché la definizione dello stile fa uso del carattere slash occorre cambiare il comportamento della libreria `lustache` per l'escaping HTML che trasformerebbe il carattere / nella sequenza `/`. La modifica si imposta nel tag inserendo tre parentesi graffe di delimitazione oppure premettendo al nome del campo contenuto il carattere `&`:

```
local tmpl_opt = [[
  {{ &style_def }}
]]
```

Il secondo template, `body`, contiene l'istruzione per il disegno della circonferenza di raggio dipendente da un parametro e l'inserimento di istruzioni `\node` nelle coordinate casuali del piano x e y :

```
local tmpl_body = [[
  \draw[dotstyle] (0,0) circle{{!
    }} [radius={{ radius }}cm];
  {{#node_list}}{<|
  }} \node[dotstyle] at ({x}}, {y}}) {};
  {/node_list}}]
```

Il codice completo dello script Lua è il seguente:

```
local lustache = require "lustache"
```

```
local tmpl_env = [[{<=<% %>=>}}{<%>
%>\begin{<% env %><%> option %><%>
%><%> body %><%>
%>\end{<% env %>}}]
```

```
local tmpl_opt = [[
  {{ &style_def }}
]]
```

```
local style_def = [[
dotstyle/.style={
  circle, % the shape
  minimum size=5mm, % the size
  % the border:
  very thick,
  draw=red!50!black!50,
  % the filling:
  top color=white,
  bottom color=red!50!black!20
}]
```

```
local tmpl_body = [[
  \draw[dotstyle] (0,0) circle{{!
    }} [radius={{radius}}cm];
  {{#node_list}}{<|
  }} \node[dotstyle] at ({x}}, {y}}) {};
  {/node_list}}]
```

```
local radius = 4 -- cm
local n = 24
local node = {}
for i=1,n do
  local alpha = 2*math.pi*math.random()
  node[#node+1] = {
    x = radius*math.cos(alpha),
    y = radius*math.sin(alpha),
  }
end
```

```

doc = lustache:render(
  tmpl_env, -- main template
  { -- data
    env="tikzpicture",
    style_def = style_def,
    node_list = node,
    radius = radius, -- cm
  },
  {-- sub template
    option = tmpl_opt,
    body = tmpl_body,
  }
)
print(doc)

```

I parametri principali come il raggio della circonferenza e il numero dei cerchietti sono memorizzati in variabili nella memoria locale del programma. Anziché con un unico template, il modello di documento è rappresentato con template specializzati per la struttura testuale associata.

9 Primo caso d'uso: griglia di attività

Questo esempio mostrerà come generare il sorgente $\text{\LaTeX}$ per comporre una tabella le cui righe rappresentano lo svolgimento di attività annuali di un gruppo di persone. Ogni cella della tabella, individuata da una colonna relativa agli anni e da una riga relativa alla persona, conterrà una barra il cui colore rappresenterà un'attività. La tabella 3 mostra un'istanza specifica di tale griglia.

L'insieme dei dati reali — fornito da Francesco Endrici — è stato opportunamente modificato sostituendo nomi e definizioni senza per questo alterare la struttura del problema. Per tradurli nella griglia utilizzeremo Lua e le sue tecniche di elaborazione delle stringhe sfruttando il costruttore delle tabelle — l'unica struttura dati di base prevista nel linguaggio — come sintassi formale per descrivere le informazioni.

9.1 Data description

Una tabella di Lua è un *dizionario*, una struttura dati formata da chiavi univoche a cui è associato un valore che può essere a sua volta una tabella. Questo tipo non solo è un contenitore molto flessibile ma ammette una sintassi letterale ispirata a $\text{\BibTeX}$ dove le chiavi stringa compaiono senza delimitatori. Questa particolarità consente al costruttore di tabelle di Lua di essere utilizzato anche come formato di descrizione dati strutturato ed espressivo. Un file di testo prodotto dall'utente tramite un editor può diventare così una sorta d'archivio con i seguenti vantaggi:

- semplice e immediata modalità d'inserimento e modifica dei dati da parte dell'utente;
- livello arbitrario di complessità;

- valutazione diretta di espressioni letterali;
- possibilità d'inserire dei commenti;
- direttamente interpretabile da Lua senza alcuna necessità di realizzare un parser per l'elaborazione successiva.

La struttura può essere disegnata per esprimere quella del problema nei termini in cui l'utente *pensa* alle informazioni. Tornando all'esempio — confronta la riga 3 della tabella 3 — il fatto che una persona di nome Sherri Castro ha svolto l'attività A nel 1972 e dal 1974 al 1976 potrebbe essere espresso scrivendo:

```
name: Sherri Castro, A: 1972, 1974->1976
```

In lua la tabella corrispondente alla sintassi letterale del costruttore sarebbe:

```
{name="Sherri Castro", A={1972,{1974,1976}}}
```

che a parole è descritta come una tabella con due chiavi, una per il nome e una per l'attività a cui è associata una seconda tabella che elenca anni e intervalli di anni (a loro volta tabelle).

Siamo liberi di indentare questa struttura per rappresentarla più chiaramente così:

```

{
  name = "Sherri Castro",
  A = { 1972,
        {1974,1976}, -- triennio
      }
}

```

Modifichiamo la tabella nella versione definitiva per poter assegnare al membro del gruppo lo svolgimento di più attività: basta introdurre una tabella per un ulteriore livello di annidamento che associamo alla chiave `task`. In questo modo si rimuove anche la limitazione della precedente struttura per cui il codice delle attività non poteva essere identificato con la chiave `name`:

```

{
  name = "Helen Austin",
  task = {-- cfr. riga 6 tabella
    A = {1977, {1983, 1985}},
    B = {{1973,1976}, 1981}
  }
}

```

9.2 Caricamento dei dati

Abbiamo due modalità per leggere le tabelle Lua: la prima è quella d'inserire i dati in un file dove un'istruzione `return` è seguita da una tabella che contiene tutte le altre e poi caricare il file nello script tramite la funzione predefinita `dofile()` che ne interpreta il contenuto come codice Lua. La seconda è premettere alle definizioni di ciascuna tabella un nome. Questa sintassi verrà interpretata da Lua come la richiesta dell'esecuzione della

TABELLA 3: Porzione della tabella di grandi dimensioni che rappresenta il grafico dello svolgimento di varie attività annuali di un gruppo di persone, prodotta elaborando un file di testo contenente i dati espressi nel formato di *data description* di Lua. La tabella originale comprende le attività di circa 160 persone nell'arco di tempo di circa 65 anni.

	1972	1973	1974	1975	1976	1977	1978	1981	1983	1984	1985	
Lisa Green	■	■	■	■		■	■					Lisa Green
Carla Figueroa	■											Carla Figueroa
Sherri Castro	■		■	■	■							Sherri Castro
Faye Ramsey		■	■									Faye Ramsey
Greg Frazier		■	■	■	■	■						Greg Frazier
Helen Austin		■	■	■	■	■		■	■	■	■	Helen Austin

funzione con lo stesso nome e con unico argomento la tabella stessa.¹³ Nello script dovremo prima definire la funzione e poi caricare il file dati ancora tramite `dofile()`.

Poiché la sua sintassi è più significativa, useremo questo secondo modo. Il seguente è un frammento del file dati a cui corrisponde la tabella 3 in cui la funzione è stata chiamata `service()`:

```

service{name = "Lisa Green",
  task = {
    ASCII = {{1972,1974}},
    EG = {1975, {1977,1978}},
  }
}

service{name = "Carla Figueroa",
  task = {
    AsE = {1972},
  }
}

service{name = "Sherri Castro",
  task = {
    ASCII = {1972,1974},
    EG = {{1975,1976}},
  }
}

service{name = "Faye Ramsey",
  task = {
    ASCII = {{1973,1974}},
  }
}

service{name = "Greg Frazier",
  task = {
    AsE = {{1973,1976}},
  }
}

service{name = "Helen Austin",
  task = {
    EG = {1977, {1983,1985}},
    LC = {{1973,1976},1981},
  }
}

```

13. In Lua le parentesi tonde della chiamata di funzione possono essere omesse se l'argomento è unico e se questo è di tipo stringa oppure di tipo tabella.

La virgola dopo l'ultimo elemento di una tabella è opzionale; inserirla facilita l'aggiunta di eventuali altri elementi, perciò considereremo il suo inserimento una buona prassi. Nel codice precedente sarebbe anche preferibile, per lo stesso motivo, dedicare una riga per ciascun anno o per ciascun intervallo di anni d'attività.

9.3 Il codice di lettura

Con la sola funzione `service()` non è possibile costruire il codice LATEX della riga di griglia. Per farlo è necessario conoscere *tutti* gli anni di colonna per poter inserire il numero corretto di separatori di cella. Il compito di questa funzione sarà allora quello di memorizzare i dati in alcune variabili globali per la successiva elaborazione d'insieme.

Per semplicità abbiamo scelto di non includere nel codice controlli di validità, per esempio per avvertire l'utente che nei dati relativi a una stessa attività un anno è stato specificato più volte o che per uno stesso anno la stessa persona ha più attività.

Poiché gli anni possono essere espressi come valori singoli o come intervalli, introduciamo la funzione ausiliaria `expand_year()` che linearizza gli intervalli e restituisce l'array dei valori semplici. La funzione `service()` mapperà gli anni in un dizionario a valori booleani per tener traccia delle colonne della griglia e aggiungerà una riga associando gli anni con le attività compiute dalla singola persona:

```

-- data section

local year_map = {}
local row_data = {}

-- expand the years tab in a plain array
local function expand_year(t_acty)
  local y_list = {}
  for _, elem in ipairs(t_acty) do
    if type(elem) == "table" then
      for y = elem[1], elem[2] do
        y_list[#y_list+1] = y
      end
    else
      y_list[#y_list+1] = elem
    end
  end
end

```

```

    return y_list
end

function service(t)
    local row = {name = t.name, years = {}}
    for acty, t_years in pairs(t.task) do
        for _, y in
            ipairs(expand_year(t_years)) do
                -- mapping global columns
                year_map[y] = year_map[y] or true
                -- insert activity
                row[y] = acty
            end
        end
        row_data[#row_data+1] = row
    end
end

```

L'effetto di queste definizioni e la conseguente esecuzione del seguente frammento di codice scritto dall'utente:

```

service{name = "Helen Austin",
  task = {
    EG = {1977, {1983,1985},},
    LC = {{1973,1976},1981,},
  }
}

```

sarà l'aggiunta in coda alle altre righe della seguente tabella Lua contenente il nome che andrà posizionato nella prima e nell'ultima colonna della griglia e anno per anno tutte le attività:

```

{name = "Helen Austin",
  years = {
    [1977] = "EG",
    [1983] = "EG",
    [1984] = "EG",
    [1985] = "EG",
    [1973] = "LC",
    [1974] = "LC",
    [1975] = "LC",
    [1976] = "LC",
    [1981] = "LC",
  }}

```

La mappa globale degli anni/colonna aggiornata per ogni esecuzione dalla funzione `service()` dovrà integrare questi dati. Nella prossima sezione vedremo infatti come la funzione `render_data()` tradurrà questa stessa tabella Lua per Helen Austin in una riga della tabella 3 nel seguente codice $\text{\LaTeX}$ qui suddiviso su due linee:

```

Helen Austin & \LC & \LC & \LC & \LC & \EG
& & \LC & \EG & \EG & \EG & Helen Austin\

```

9.4 Il codice di elaborazione

Al termine dell'esecuzione di tutte le funzioni `service()` contenute nel file dati, saranno disponibili in memoria tutte le informazioni necessarie alla creazione della griglia in due tabelle Lua: la mappa degli anni di attività per tutte le persone e l'array con i dati delle righe. Passando le relative variabili alla funzione `render_data()` produrremo

un file contenente un ambiente `tabular` che sarà a sua volta inserito in un sorgente $\text{\LaTeX}$ principale, basato sulla classe `standalone`, che definirà i colori, le dimensioni, i font, le macro e caricherà i pacchetti necessari.

In particolare, faremo uso del pacchetto `array` per ottenere colonne centrate e della stessa larghezza, il pacchetto `booktabs` per i filetti orizzontali della prima e dell'ultima riga, il pacchetto `xcolor` per definire i colori delle barre nelle celle e il pacchetto `colortbl` per assegnare il colore dei filetti.

Il contenuto delle celle non vuote sarà una barra colorata orizzontale prodotta da una macro il cui nome corrisponderà a quello dei codici identificativi delle attività usati nei dati. Per esempio, all'interno del sorgente principale la definizione delle grandezze dimensionali in sintassi $\text{\LaTeX}$ comuni a tutte le barre, la definizione del colore e della macro per l'attività "EG" sarà:

```

% definition of the bar colors
...
\definecolor{dep}{rgb}{0.24,0.55,0.20}
...

% column width: 1.5 the length of ``2012''
\newlength{\tcellwidth}
\settowidth{\tcellwidth}{2012}
\setlength{\tcellwidth}{1.5\tcellwidth}

% definition of the bar dimensions
\newlength{\tbarheight}
\setlength{\tbarheight}{5pt}
\newlength{\tbarwidth}
\setlength{\tbarwidth}{0.97\tcellwidth}
\newlength{\tbarup}
\setlength{\tbarup}{0.5pt}

% the bar 'factory' command
\newcommand{\servicebar}[1]{\{
\color{#1}%
\rule[\tbarup]{\tbarwidth}{\tbarheight}%
}}

% the bar commands
...
\newcommand{\EG}{\servicebar{dep}}
...

```

Tornando a Lua, la funzione `render_data()` utilizzerà la mappa degli anni/colonna per generare la definizione delle colonne, parametro obbligatorio dell'ambiente `tabular`, e la prima riga d'intestazione, mentre utilizzerà i dati delle righe per generare la sequenza di celle separate dal carattere '&'. Ancora una volta il processo memorizzerà l'intero ambiente `tabular` riga per riga in una tabella Lua che verrà poi unita nel testo finale in modo efficiente con la funzione predefinita `table.concat()`.

Il codice completo è riportato di seguito. Esso non è né semplice né elegante perché disseminato di stringhe di codice $\text{\LaTeX}$ che lo rendono poco leggibile. Utilizzare le tecniche basate sulle stringhe

era del resto una scelta iniziale del progetto ma lasciamo al lettore per utile esercizio riscrivere il codice questa volta utilizzando il template engine *lustache*.

```
-- rendering data section

function render_data(t_years, t_data)
  -- ordered list of column years
  local col_years = {}
  for y in pairs(t_years) do
    col_years[#col_years+1] = y
  end
  table.sort(col_years)

  local rs = {}
  -- tabular header
  local cdef = "r@{\hspace{0.5em}}"..
    "*{%d}{@}C{\tcellwidth{@}}"..
    "@{\hspace{0.5em}}l"
  rs[#rs+1] = string.format(
    "\\begin{tabular}{%s}",
    string.format(cdef, #col_years)
  )
  rs[#rs+1] = "\\toprule"
  local colfmt = {}
  for _, y in ipairs(col_years) do
    colfmt[#colfmt+1] = string.format(
      "\\textbf{%d}",
      y
    )
  end
  rs[#rs+1] = "& " ..
    table.concat(colfmt, " & ") ..
    " &\\\\"
  rs[#rs+1] = "\\midrule"

  -- tabular body
  for i, row in ipairs(t_data) do
    local rndrow = {}
    for _, year in ipairs(col_years) do
      if row[year] then
        rndrow[#rndrow+1] = "\\\" ..
          row[year]
      else
        rndrow[#rndrow+1] = " "
      end
    end
    local hline
    if i == #t_data then
      hline = ""
    else
      hline = "\\hline"
    end
    rs[#rs+1] = row.name ..
      " & " ..
      table.concat(rndrow, " & ") ..
      " & " ..
      row.name ..
      "\\\" ..
      hline
  end

  -- tabular ending
```

```
rs[#rs+1] = "\\bottomrule"
rs[#rs+1] = "\\end{tabular}"
return rs
end
```

9.5 Il codice di scrittura

Non rimane che completare lo script scrivendo il codice che carica il file dati, esegue la funzione di traduzione nel codice L^AT_EX e salva il risultato in un file esterno:

```
-- execution section

dofile("services.lua")
local t = render_data(year_map, row_data)
-- save the LaTeX file
local tex = io.open("tabdata.tex", "w")
tex:write(table.concat(t, "\n"))
tex:close()
```

9.6 Considerazioni finali sulla griglia

Rispetto alla soluzione originale scritta in L^AT_EX3 quella presentata qui in Lua non è tanto più efficiente quanto è più leggibile e robusta e ciò porta a maggiori potenzialità di miglioramento per effetto dell'uso di un linguaggio ad alto livello progettato espressamente anche per questi compiti.

Per esempio, non sarebbe difficile modificare il programma per considerare anche il caso in cui una persona svolga più attività in uno stesso anno, oppure per eseguire l'ordinamento temporale delle righe della griglia evitando all'utente l'obbligo di inserire in sequenza le tabelle dati *service*, oppure per creare *griglie finestra* selezionando un intervallo di anni.

Mentre il formato di dati originale è un file *.csv* (*comma separated values*) proveniente da un foglio di calcolo, in questo esempio applicativo le stesse informazioni sono state tradotte in un formato di *data description* direttamente interpretabile da Lua sfruttando la flessibile sintassi del suo costruttore di tabelle. Con esso è stata progettata una struttura espressiva pensando alle informazioni inerenti allo svolgimento di attività annuali da parte di componenti di un gruppo che l'utente può inserire manualmente.

10 Python

L'ultimo linguaggio considerato in questo lavoro è Python, diffusissimo linguaggio di scripting per molto tempo contrapposto a Perl.

10.1 Stringhe e campi

In Python3 le stringhe sono codificate Unicode *utf-8* perciò preferiamo questa versione — in particolare la 3.4 — rispetto a quella della precedente serie stabile. Al pari di Lua, il linguaggio prevede diverse sintassi per inserire valori stringa letterali usando due diversi delimitatori: gli apici semplici o doppi per le stringhe di un'unica riga oppure tre

apici semplici o doppi consecutivi per le stringhe multiriga. Premettendo il modificatore `r` (*raw*) si disinseriscono le sequenze di escape, cosa utile, per esempio, nei casi di stringhe contenenti il carattere backslash delle macro `TeX`.

Nelle prossime sezioni è riportato del codice che fa uso della sintassi per le stringhe grezze sia mono che multiriga e della funzione `format()` operante come metodo del tipo stringa per testi con campi variabili.

10.2 Costruire una tabella

Diversamente dagli esempi precedenti, nel prossimo script il testo `LaTeX` della tabella numerica già realizzata in Lua non verrà stampato a video sul canale di uscita ma sarà memorizzato in un file su disco. Questo ci darà modo di presentare le istruzioni necessarie per accedere al file system.

```
# creating a new list also called 't'
t = [r"\begin{tabular}{crrrr}"]
t.append(r"\toprule")
t.append(r"\(x\) & \(\x^2\) & \(\x^3\)")
t.append(r"      r" & \(\x^4\) & \(\x^5\)\\")
t.append(r"\midrule")

f = (r" {0:2d} & {1:3d} & {2:4d} & "
     r" {3:5d} & {4:6d}\\")
for x in range(1,11):
    t.append(
        f.format(x, x**2, x**3,
                 x**4, x**5
        ))

t.append(r"\bottomrule")
t.append(r"\end{tabular}")

# saving content in the TeX file
fl = open("test-table.tex", "w")
fl.write("\n".join(t))
fl.close()
```

Nonostante le molte differenze concettuali tra Lua e Python, i due programmi per produrre la tabella numerica d'esempio sono molto simili. Nel caso di Python si usa una *lista* allo stesso modo della tabella/array in Lua, il metodo `format()` al posto della funzione `string.format()` e il metodo `join()` al posto della funzione `table.concat()`.

La figura 6 mostra l'editor IDLE, dedicato alla programmazione in Python, al lavoro sul codice del programma appena presentato.

10.3 Jinja un template engine per Python

Per proseguire nello studio della tecnologia dei template con il linguaggio Python abbiamo scelto il modulo `Jinja2` — versione 2.7.2 per Python3; la figura 7 ne mostra il logo — perché molto ricco di funzionalità; tra esse abbiamo i delimitatori adattabili per la generazione di sorgenti `LaTeX` e la precompilazione dei template per incrementare le prestazioni.

La prima differenza che incontriamo rispetto a `lustache` è che i tag dei template non hanno un carattere marcatore che ne specifica il tipo — come il `#` per un blocco o il `!` per un commento — ma delimitatori diversi per ciascun tipo. I tipi di tag in `Jinja` sono quattro e sono qui elencati con i delimitatori di default:

- `{% ... %}` istruzioni
- `{{ ... }}` espressioni valutate e inserite nel testo finale,
- `{# ... #}` commenti non inclusi nell'uscita del template,
- `# ... ##` istruzioni in linea.

10.4 Merging PDF file

Come primo esempio con il modulo `Jinja` studieremo la creazione di un file sorgente `LaTeX` per l'unione di più file PDF tramite il pacchetto `pdfpages`.

Nel template troveremo un tag istruzione per un ciclo iterativo che genererà la sequenza di macro `\includepdf` con una sintassi molto simile a quella in Python. La variabile d'iterazione è inserita in un tag espressione con delimitatori personalizzati:

```
tmpl = r"""
\documentclass{minimal}
\usepackage{pdfpages}
\begin{document}
{% for doc in doc_list -%}
    \includepdf[pages=-]{<<| doc |>>}
{% endfor -%}
\end{document}
"""
```

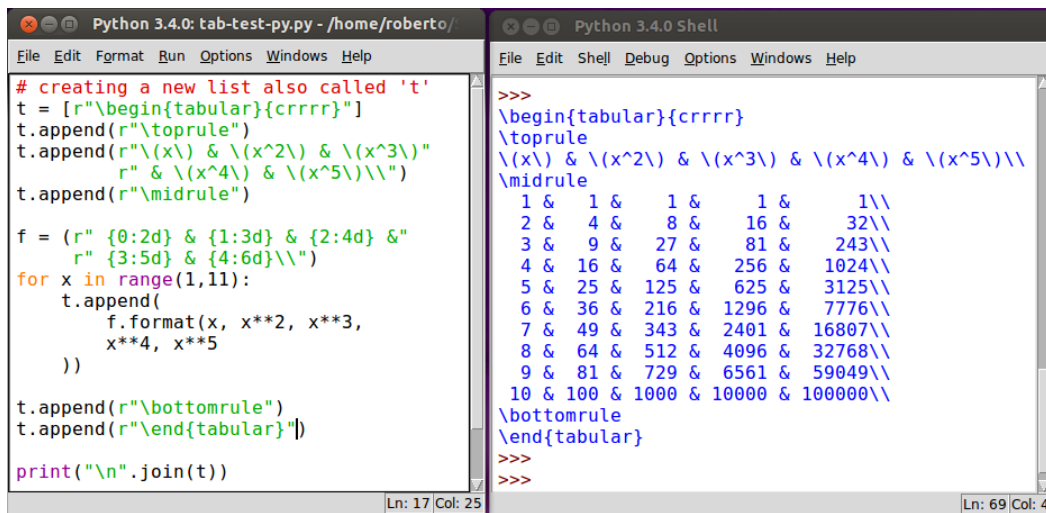
Il template è molto più leggibile che non l'equivalente per `lustache` perché possiamo inserire prima del tag di chiusura il modificatore trattino -, che comporta l'eliminazione dei caratteri spazio e di ritorno a capo dal testo finale successivi al delimitatore di chiusura del tag. Questo comportamento può essere reso implicito configurando opportunamente gli oggetti restituiti dal modulo.

Veniamo all'esame del codice dello script: per prima cosa si carica l'oggetto `Environment` del modulo `jinja2` istanziandone l'oggetto con il costruttore a cui vengono passate le nuove stringhe di delimitazione dei tag di tipo *variable*. La modifica dei delimitatori non è quindi locale al template ma globale nell'ambiente.

Il template binario è ottenuto tramite il metodo `from_string()` dell'ambiente mentre il testo finale viene generato chiamando il metodo `render()` con la tupla dei nomi dei file da unificare:

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

from jinja2 import Environment
```



```

Python 3.4.0: tab-test-py.py - /home/roberto/
File Edit Format Run Options Windows Help
# creating a new list also called 't'
t = [r"\begin{tabular}{crrrr}"]
t.append(r"\toprule")
t.append(r"\(x\) & \((x^2)\) & \((x^3)\) & \((x^4)\) & \((x^5)\)")
t.append(r"\midrule")

f = (r" {0:2d} & {1:3d} & {2:4d} & {3:5d} & {4:6d}")
for x in range(1,11):
    t.append(
        f.format(x, x**2, x**3, x**4, x**5)
    )

t.append(r"\bottomrule")
t.append(r"\end{tabular}")

print("\n".join(t))
Ln: 17 Col: 25

Python 3.4.0 Shell
File Edit Shell Debug Options Windows Help
>>>
\begin{tabular}{crrrr}
\toprule
\((x\) & \((x^2\) & \((x^3\) & \((x^4\) & \((x^5\)\\
\midrule
1 & 1 & 1 & 1 & 1\\
2 & 4 & 8 & 16 & 32\\
3 & 9 & 27 & 81 & 243\\
4 & 16 & 64 & 256 & 1024\\
5 & 25 & 125 & 625 & 3125\\
6 & 36 & 216 & 1296 & 7776\\
7 & 49 & 343 & 2401 & 16807\\
8 & 64 & 512 & 4096 & 32768\\
9 & 81 & 729 & 6561 & 59049\\
10 & 100 & 1000 & 10000 & 100000\\
\bottomrule
\end{tabular}
>>>
>>>
Ln: 69 Col: 4

```

FIGURA 6: L'editor IDLE in azione mostra una finestra principale con il codice dello script in Python e una finestra di shell interattiva che raccoglie l'uscita dei programmi. Per eseguire il codice è sufficiente premere il tasto F5.



FIGURA 7: Il logo del progetto Jinja rappresenta un tempio giapponese per l'assonanza tra i termini 'template' e 'temple'.

```

env = Environment(
    variable_start_string="<<|",
    variable_end_string="|>>"
)

tmpl = r"""
\documentclass{minimal}
\usepackage{pdfpages}
\begin{document}
{% for doc in doc_list -%}
    \includepdf[pages=-]{<<|doc|>>}
{% endfor -%}
\end{document}
"""

t_eng = env.from_string(tmpl)

docs = ("as541","as656","as163",
        "as048","as525","as854")
print(t_eng.render(doc_list=docs))

```

10.5 Pipeline e filtri

In Jinja esiste il concetto di *pipeline* di comandi: l'uscita di un comando verrà passata come ingresso

alla funzione successiva separata da un carattere | che simboleggia un condotto. Questa funzionalità è chiamata *filtro*, una scelta non particolarmente felice perché essa è un'operazione di mappatura e non di filtraggio.

Come esempio, nel prossimo script applicheremo un filtro al titolo di sezione per renderlo in caratteri maiuscoli e per arrotondare un numero decimale a sei cifre significative:

```

from jinja2 import Environment
import math

env = Environment(
    variable_start_string="<<|",
    variable_end_string="|>>"
)

tmpl = r"""
\section{<<| sec_tit|upper |>>}

A rounded version of the \(\pi\) value:
<<| "%0.6f"|format(pi) |>>.
"""

t_eng = env.from_string(tmpl)
print(t_eng.render(sec_tit="Important",
    pi = math.pi))

```

Si tratta di dettagli di *presentazione* dei dati perciò il template non viola troppo la regola fondamentale di separazione tra modello e vista — in questo caso tra dati e testo finale — includendone la logica. Le operazioni di elaborazione dovranno riguardare esclusivamente proprietà di visualizzazione e mai la costruzione dei dati. In altre parole, al template è deputata la responsabilità di presentare i dati elaborandoli opportunamente.

Nell'esempio precedente il template non conosce il valore di π e si occupa solamente di arrotondare un numero decimale.

TABELLA 4: Una semplice tabella esaminata nel testo come vista sui dati. Il template si occupa dei dettagli come la numerazione delle righe e il calcolo del totale di colonna.

N.	Description	Cost
1	Site preparation	72030.00
2	Excavation work	242000.00
3	Foundations	5078000.00
4	Main framework	8925000.00
Total		14317030.00

10.6 Materiale tabellare

Consideriamo la tabella 4 come *vista* sui dati. Vorremmo che il template si occupasse della formattazione dei numeri a due posizioni decimali fisse, del calcolo del totale di colonna e della numerazione automatica delle righe.

Tutto ciò è piuttosto semplice in Jinja grazie ad alcune funzioni predefinite. Abbiamo già incontrato la funzione `format()` messa in pipeline con la stringa di formato. Un'altra utile funzione è `sum()`, che riceve una lista per sommarne gli elementi e opzionalmente anche il nome dell'attributo associato al valore da considerare nel calcolo se il dato è, per esempio, un dizionario.

Per la numerazione delle righe è sufficiente usare l'oggetto `loop` che assieme a molte altre informazioni contiene il contatore delle iterazioni nel campo `index`.

Il manuale di riferimento online del *template engine* presenta tutte le funzioni con esempi di codice e non è difficile trovare proprio la funzione necessaria a risolvere un dato problema. Di seguito il codice completo dello script che genera la tabella 4.

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

from jinja2 import Environment
env = Environment()

tmpl = r"""
\begin{tabular}{crr}
\toprule
N. & Description & Cost\\
\midrule
{% for w in data_list -%}
    {{ loop.index -}}
    & {{ w.descr -}}
    & {{ "%0.2f"|format(w.cost) }}\\
{% endfor -%}
\midrule
    & Total & {{
        "%0.2f"
        |format(
            data_list
            |sum(attribute='val')) }}\\
\bottomrule
\end{tabular}
"""
```

```
t_eng = env.from_string(tmpl)
data = (
    {'descr': "Site preparation", 'val': 72030},
    {'descr': "Excavation work", 'val': 242000},
    {'descr': "Foundations", 'val': 5078000},
    {'descr': "Main framework", 'val': 8925000},
)
print(t_eng.render(data_list=data))
```

Nel listato 1 è riportato il codice esteso dello script precedente in cui viene aggiunta una colonna i cui valori sono le percentuali date dei valori corrispondenti di quella dei costi.

La particolarità sta nel calcolo della somma di queste percentuali a opera del template: nonostante la possibilità di utilizzare variabili tramite il tag `set`, non è possibile leggerne il valore modificato all'interno del ciclo `for` perché gli incrementi riga per riga rimangono *locali* al blocco.

Abbiamo quindi utilizzato la lista `vatlist`, gestita in memoria come oggetto globale, appendendovi i singoli valori all'interno di un blocco `if` vuoto. All'esterno del ciclo che produce le righe della tabella questa lista viene poi passata alla funzione `sum()` che calcola il totale desiderato.

Se nel modulo fosse previsto un tag `global set`, il codice ritornerebbe a essere quello che ci si aspetterebbe ma probabilmente le variabili globali invoglierebbero a inserire la logica dati nel template.

10.7 Funzioni custom

In questo esempio vedremo come utilizzare nel template le funzioni personalizzate. Lo scopo è, lo ricordiamo ancora, modificare solo l'aspetto dei dati per crearne una *vista*.

Lo script costruirà il codice L^AT_EX della seguente tabella composta da una prima colonna con i numeri non formattati e da una seconda colonna con i numeri formattati con la nuova funzione personalizzata.

No formatting	Formatting
45613854.156	45 613 854,16
546447.1	546 447,10
450215	450 215,00
454	454,00
7899.3	7899,30

Per poter chiamare una funzione in un tag istruzione del template è sufficiente registrarla nel dizionario `globals` dell'oggetto `Environment`. La chiave dovrà coincidere con il nome della funzione utilizzata internamente nel template.

Nello script si procede col definire una semplice funzione di formato numerico a precisione fissa a due decimali, che inserisce un piccolo spazio tra i gruppi di tre cifre se la parte intera del numero ne ha almeno cinque, secondo le norme del Sistema

LISTATO 1: Il codice dello script che genera una tabella compiendo calcoli percentuali e somme sui dati.

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

from jinja2 import Environment

env = Environment()

tmpl = r"""
\begin{tabular}{clrr}
\toprule
N. & Description & Cost & \textsc{vat}\\
\midrule
{% set vatlist = [] -%}
{% for w in data_list -%}
    {% just a trick...
    #}{% if vatlist.append(w.cost*w.vat) -%}{%endif -%}
    {{ loop.index
    & {{ "%-18s"|format(w.descr) -}}
    & {{ "%10.2f"|format(w.cost) -}}
    & {{ "%8.2f"|format(w.cost*w.vat) }}\\
{% endfor -%}
\midrule
& Total & {{
"%0.2f"
|format(data_list|sum(attribute='cost')) -}}
& {{ "%0.2f"|format(vatlist|sum) }}\\
\bottomrule
\end{tabular}
"""

t_eng = env.from_string(tmpl)
data = (
    {'descr': "Site preparation", 'cost': 720, 'vat': .10},
    {'descr': "Excavation work", 'cost': 2420, 'vat': .16},
    {'descr': "Foundations", 'cost': 50780, 'vat': .16},
    {'descr': "Main framework", 'cost': 89250, 'vat': .16},
)
print(t_eng.render(data_list=data))

# program's output:
'''

\begin{tabular}{clrr}
\toprule
N. & Description & Cost & \textsc{vat}\\
\midrule
1& Site preparation & 720.00 & 72.00\\
2& Excavation work & 2420.00 & 387.20\\
3& Foundations & 50780.00 & 8124.80\\
4& Main framework & 89250.00 & 14280.00\\
\midrule
& Total & 143170.00 & 22864.00\\
\bottomrule
\end{tabular}
'''
```

Internazionale delle unità di misura.¹⁴ La funzione tiene conto anche di un parametro opzionale che controlla il separatore decimale. Il codice completo dello script è il seguente:

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

from jinja2 import Environment

def fmt_num(x, comma_sep=False):
    if int(x) > 9999:
        res = "{:,.2f}".format(x)
        res = res.replace(",", "\,")
    else:
        res = "{:.2f}".format(x)

    if comma_sep:
        return res.replace(".", ",")
    else:
        return res

env = Environment()
env.globals['fmt_num'] = fmt_num

tmpl = r"""
\begin{tabular}{rr}
\toprule
No formatting & Formatting\\
\midrule
{% for x in nums -%}
    {{x}} & {{ fmt_num(x, True) }}\\
{% endfor -%}
\bottomrule
\end{tabular}
"""

nums = (45613854.156,
        546447.1,
        450215,
        454,
        7899.3)

t_eng = env.from_string(tmpl)

print(t_eng.render(nums=nums))
```

10.8 Template ricorsivi

L'effetto è sostanzialmente simile a quello dei template figli di *lustache* esaminati alla sezione precedente: in un template possono essere inclusi altri template secondari per mezzo del tag `include`. La struttura che ne deriva è una gerarchia ad albero dall'alto verso il basso che corrisponde a un unico documento strutturato in parti e sottoparti.

Poiché i sorgenti L^AT_EX sono composti da parti e sottoparti questa funzionalità è particolarmente utile. Un esempio di strutturazione dei template è riportato nella sezione 11.

14. Ulteriori informazioni possono essere trovate nella documentazione del pacchetto *siunitx* oppure nella pagina web <http://physics.nist.gov/cuu/Units/checklist.html> al punto 16.

10.9 Ereditarietà dei template

L'ereditarietà è uno dei concetti più importanti della programmazione a oggetti. Consente di creare una classe contenente le funzionalità di base comuni a una famiglia di oggetti sempre più specializzati. Nel caso dei modelli di documento un template di base può essere esteso da un nuovo template indipendente.

Se desideriamo produrre documenti diversi che tuttavia abbiano in comune una serie di elementi, non potremo riusare il codice con i template ricorsivi perché l'inclusione dei moduli secondari sarà riferita a un template fisso e quindi a sempre e solo un unico documento. La ricorsione esprime cioè la struttura di un unico documento mentre l'ereditarietà rappresenta la struttura di una *famiglia* di documenti.

Nel template base si troverà almeno un tag di tipo istruzione di nome `block` che verrà poi, per così dire, *implementato* o *esteso* da un secondo template. Per esempio:

```
# base template:
t_m = r"""
\begin{document}
{% block content -%}
{% endblock -%}
\end{document}
"""
```

Il template che eredita un template base dichiarerà di estenderlo con il tag di nome `extends` e poi definirà i blocchi base. Per esempio:

```
# child template
t_c = r"""
{% extends "main" -%}
{% block content -%}
Great {{ thing }}!
{% endblock -%}
"""
```

L'ereditarietà in Jinja è una funzionalità molto potente. Per esempio anche un singolo blocco può essere esteso chiamando la funzione `super()` o può essere stampato più di una volta tramite la chiave `self.<block-name>()` oppure ancora può essere inserito in un ciclo iterativo. Vediamo brevemente un esempio di ciascuna di queste funzionalità.

La parola chiave *super()*

La dimensione della pagina di una famiglia di documenti basati sulla classe `article` può essere fissata tra le opzioni nel template di base; tali opzioni potranno essere estese nel template derivato come in questo codice:

```
# base template
t_m = r"""
\documentclass[{{ block option -%}}
a4paper{% endblock %}]{article}
"""
```

```
t_c = """\n
{% extends "main" -%}
{% block option -%}
{{ super() -%}}
{% if opt %},{% endif -%}
    {{ opt|join(',') -%}}
{% endblock -%}
"""
```

Se la lista `opt` contiene le stringhe `'12pt'` e `'draft'` il risultato sarà:

```
\documentclass[a4paper,12pt,draft]{article}
```

La parola chiave `self.<block-name>()`

Con lo stesso esempio dimostriamo la capacità d’inserire copie del testo di un blocco con la funzione `self.<block-name>()` riportata nel template di base:

```
# base template
t_m = r"""
\documentclass[{{% block option -%}}]{article}
a4paper{% endblock %}}{article}
% le opzioni di classe sono:
% {{ self.option() }}
"""
```

e il risultato sarà:

```
\documentclass[a4paper,12pt,draft]{article}
% le opzioni di classe sono:
% a4paper,12pt,draft
```

Blocchi innestati e la parola chiave `scoped`

Nel template di base un blocco è incluso all’interno di un ciclo iterativo, per esempio perché contiene il testo di una riga di una tabella composta con l’ambiente `tabular`. Perché le variabili di ciclo siano visibili all’interno del blocco è necessario specificare la chiave `scoped` per modificare il comportamento *difensivo* del template:

```
t_m = r"""
\begin{tabular}{cc}
{% for key, value in mydict.items() -%}
{% block row scoped -%}{{% endblock -%}}
{% endfor -%}
\end{tabular}
"""

t_c = """\n
{% extends "main" -%}
{% block row -%}
    {{key}} & {{value}}\\\\\n
{% endblock -%}
"""
```

Se il dizionario contiene le seguenti informazioni `{'font':'12pt', 'paper':'a4paper', 'version':'draft'}` il risultato sarà:

```
\begin{tabular}{cc}
paper & a4paper\\
version & draft\\
font & 12pt\\
\end{tabular}
```

10.10 Un esempio di ereditarietà

Memorizzando i template in file esterni avremo il vantaggio che le modifiche non interessanti un cambiamento della struttura dei dati non riguarderanno l’applicazione ma solamente questi file di testo. Negli esempi di questa sezione i template sono inclusi nel sorgente per scopi esplicativi e continueremo a farlo anche se sarà necessario introdurre funzioni diverse da quelle che avremmo utilizzato in un’applicazione reale.

In Jinja la sequenza di passi effettiva infatti è quella di utilizzare un apposito *loader* per leggere i template da disco — anche in forma precompilata. Questo è il motivo per cui il metodo `from_string()`, usato fino a ora negli esempi per generare oggetti `Template`, non prevede che i template possano rappresentare una gerarchia.

Tuttavia non è obbligatorio che i template si trovino memorizzati nel file system perché il modulo `Jinja` mette a disposizione l’oggetto `DictLoader` che elabora un dizionario e restituisce il loader con cui caricare le stringhe del template base e del template d’estensione con i nomi associati.

C’è da notare che il metodo `get_template()` dovrà richiedere all’oggetto `Environment` il template figlio e non quello base perché è il primo che ne eredita ed estende il contenuto.

Nell’esempio presentato il risultato atteso dei template sono i due sorgenti LATEX che hanno lo stesso preambolo ma un diverso corpo:

```
\documentclass{article}
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage[italian]{babel}
\begin{document}
Great job!
\end{document}

\documentclass{article}
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage[italian]{babel}
\begin{document}
Hello Albert!
\end{document}
```

Il codice completo dello script che li realizza è il seguente:

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

from jinja2 import Environment, DictLoader

t_m = r"""
\documentclass{article}
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage[italian]{babel}
\begin{document}
{% block content -%}
{% endblock -%}
```

```

\end{document}
"""

t_c1 = """\
{% extends "main" -%}
{% block content -%}
Great {{ thing }}!
{% endblock -%}
"""

t_c2 = """\
{% extends "main" -%}
{% block content -%}
Hello {{ name }}!
{% endblock -%}
"""

loader = DictLoader(
    {'main':t_m,
     'content1':t_c1,
     'content2':t_c2}
)
env = Environment(loader=loader)

tpl1 = env.get_template('content1')
print(tpl1.render(thing="job"))

tpl2 = env.get_template('content2')
print(tpl2.render(name='Albert'))

```

11 Secondo caso d'uso: l'inventario

In questo esempio reale mostreremo come produrre un report d'inventario di un esercizio commerciale interrogando un database relazionale. I dati sono memorizzati in un database SQLite3 (SQLite3). Libreria libera estremamente diffusa e collaudata, progettata per l'utilizzo efficiente su postazioni locali, SQLite3 non si basa sulla classica architettura *client/server* distribuita su una rete e non c'è quindi la necessità di effettuare alcuna configurazione d'accesso mentre è immediata la gestione dei dati salvati in un singolo, ordinario file locale. Si tratta comunque di un *database engine* transazionale ACID¹⁵ *compliant*, multiplatforma e disponibile praticamente per qualsiasi sistema operativo nella forma di un file sorgente scritto in linguaggio C per l'inclusione diretta in applicazioni o in librerie a caricamento dinamico.

Il programma di creazione del report sarà suddiviso in due principali fasi d'esecuzione: il reperimento delle informazioni dal database e la conseguente produzione del sorgente L^AT_EX a partire dalle informazioni ottenute; in dettaglio:

1. consultazione del database SQL:
 - (a) connessione al database;
 - (b) esecuzione delle query e inserimento dati in memoria;

15. Per maggiori informazioni si consulti la pagina web <https://en.wikipedia.org/wiki/ACID>.

TABELLA 5: Tabella con dati di esempio rappresentativa dell'aspetto del report d'inventario secondo le specifiche definite nel testo.

Supplier: <i>Good Shirts</i>			
Description	Q.ty	Cost (\$)	Total (\$)
AQ12 PL21 — T-Shirt Pluton series	12	12.50	150.00
SZ02 PL24 — T-Shirt Pluton series	8	19.10	152.80
W964 JP02 — T-Shirt Jupiter series	6	12.90	77.40
BP01 JP10 — T-Shirt Jupiter series	20	15.80	316.00
XVYY E5 — T-Shirt Earth series	5	14.80	74.00
XVYY E19 — T-Shirt Earth series	3	12.20	36.60
XVYY E41 — T-Shirt Earth series	19	11.30	214.70
NEAR 956 — T-Shirt Moon series	5	14.80	74.00
	78		1095.50

(c) chiusura della connessione.

2. costruzione del sorgente L^AT_EX:

- (a) definizione delle funzioni accessorie per la presentazione dei dati;
- (b) caricamento del template;
- (c) unione dei dati;
- (d) memorizzazione del risultato su file.

Come sarà ormai chiaro, scriveremo il programma in Python3, che fornisce con i propri moduli di libreria l'accesso immediato a database SQLite3, e produrremo il sorgente con il template engine Jinja. Questa procedura è più efficiente e affidabile che non quella basata sull'utilizzo di fogli di calcolo perché l'uso di un database assicura la coerenza e l'integrità dei dati e al contempo l'evoluzione futura della struttura relazionale.

Per dare un'idea delle prestazioni del processo descritto, il tempo di esecuzione per creare il sorgente di un inventario reale di circa 8000 articoli è risultato essere di 0,17 s sulla macchina di riferimento.

11.1 Specifiche del report

L'inventario annuale è l'elenco della merce presente in magazzino alla data del 31 dicembre. Questo elenco dovrà essere suddiviso per reparto e per fornitore riportando i totali parziali sia di quantità che di costo. Al termine dell'elenco dovrà essere presentata una tabella riassuntiva per reparto per dare le proporzioni generali di consistenza del magazzino.

Nell'elenco ogni riga sarà formata dalle seguenti informazioni: un breve testo descrittivo che definisce univocamente l'articolo, la quantità inventariata, il costo del singolo pezzo e il totale risultante. L'elenco dovrà apparire su due colonne per pagina e dovrà essere presente un frontespizio.

Nella tabella 5 è riportato un breve possibile estratto dell'elenco d'inventario composto secondo le specifiche decise per il documento finale.

11.2 Gerarchia dei template

Per produrre il report dell'inventario, anziché utilizzare un unico template ne introdurremo uno

principale che definirà il preambolo del sorgente e il frontespizio e altri due sub-template, uno per inserire l'elenco degli articoli e l'altro per comporre la tabella riassuntiva finale. In questo modo è più semplice gestire e mettere a punto le singole parti rispettando la struttura effettiva del documento finale. I template saranno salvati su file e potranno essere modificati indipendentemente dal codice dello script. Faremo uso di funzionalità della libreria Jinja non ancora presentate negli esempi già visti; questi ultimi includono i template come semplici variabili stringa definite nello script.

11.3 Il template principale

Useremo la classe L $\TeX$ `report` con i pacchetti di base per impostare codifiche, dimensione della pagina, gabbia del testo, testatine, font, numero di colonne; useremo poi il pacchetto `supertabular` per comporre l'elenco degli articoli inventariati sotto forma di tabella a più pagine per ciascun reparto. Per brevità riporteremo solamente il corpo del sorgente L $\TeX$ inserito nel template principale. Dopo aver definito le righe d'intestazione e di coda previste dal pacchetto `supertabular`, troviamo le due istruzioni per l'inclusione dei sub-template tramite il nome del corrispondente file racchiuso tra doppi apici:

```
\begin{document}
\maketitle
\twocolumn

\scriptsize
\sffamily

% intestazione tabella d'inizio pagina
\tablehead{
\hline
Descrizione articolo & Q.tà & Costo \euro &
Esist. \euro\\\hline

\tabletail{% coda a fine pagina tabella
\hline
\multicolumn{4}{r}{\textit{segue\dots}}\\
}

% ultima riga tabella
\tablelasttail{\hline
\multicolumn{4}{r}{%
\textit{Fine tabella di reparto.}}\\}

<<% include "supertab-tmpl.tex" -%>>

\newpage
\onecolumn
\normalsize

<<% include "finaltable-tmpl.tex" -%>>

\bigskip
\rule[.5ex]{180pt}{1pt}
Fine documento
```

```
\rule[.5ex]{180pt}{1pt}
\end{document}
```

11.4 Il sub-template elenco

Nel seguente codice è riportato l'intero contenuto del sub-template che formerà l'elenco degli articoli secondo le specifiche del report. Tutto il testo è racchiuso in un ciclo eseguito per ogni reparto, il che significa che verranno prodotti altrettanti ambienti `supertabular`.

La funzione Python `items()`, definita per il tipo dizionario, restituisce un iteratore a due valori: il primo è la chiave e il secondo è il valore a essa associato. Corrispondentemente lo script definirà il campo `data` come un dizionario `{(nome reparto)/(dizionario fornitori)}`.

Per ogni reparto definiamo la prima riga della tabella dell'elenco e l'ambiente `supertabular` contenente un ulteriore ciclo annidato per ciascun fornitore sulle variabili `{(codice fornitore)/(lista articoli)}`.

Per ogni fornitore il ciclo di ultimo livello inserirà le righe della tabella con descrizione, quantità, costo e costo totale, a rappresentare la struttura riportata schematicamente in figura 8. Tutti i campi sono formattati per essere lunghi lo stesso numero di caratteri in modo che nel sorgente i dati risultino allineati in verticale — in particolare i valori di costo sono arrotondati a due cifre decimali — per mezzo delle funzioni `format()` e `truncate()`.

Il template prevede anche che sia definita una funzione di nome `get_sup_name()` la quale, dato il codice, restituisca il nome completo del fornitore e che i valori delle somme parziali di quantità e costi siano contenute in un dizionario strutturato per livelli `(reparto)/(fornitore)`.

```
<<% for dep_name, sup_list in
data.items()-%>>

\tablefirsthead{
\hline
\multicolumn{4}{c}{%
--- REPARTO <<| dep_name|upper |>> ---}\\
\hline
}

\begin{supertabular}{lrrr}
\\ \hline
Descrizione articolo & Q.tà & Costo \euro
& Esist. \euro\\\hline
<<% for sup_code, artlist
in sup_list.items() -%>>
\\
\multicolumn{4}{l}{%
Fornitore <<| get_sup_name(sup_code) |>>}\\
\hline
<<% for art in artlist -%>>
<<| "%-38s"|format(art.descr)
|truncate(38, end="")
|>> & <<| "%5d"|format(art.qty)
|>> & <<| "%6.2f"|format(art.cost)
|>> & <<| "%7.2f"|format(art.qty*art.cost)
```

```

|>> \\\
<<% endfor -%>>
\hline
<<| "%-38s"|format("Tot.")
|>> & <<| "%5d"|format(sums[dep_name]
                        ['suptot']
                        [sup_code]
                        ['qty'])
|>> & & <<| "%8.2f"|format(sums[dep_name]
                        ['suptot']
                        [sup_code]
                        ['tot'])

|>> \\\
\hline
<<% endfor -%>>
\\
\\
\hline
<<| "Tot. Rep. %-32s"|format(dep_name)
|>> & <<| "%5d"|format(sums[dep_name]
                        ['qtytot'])
|>> & & <<| "%8.2f"|format(sums[dep_name]
                        ['tottot'])

|>> \\\
\hline
\end{supertabular}
<<% endfor -%>>

```

11.5 Il sub-template quadro

Per costruire il quadro riassuntivo utilizziamo un ambiente `tabular` centrato nella pagina. Poiché le cifre saranno elevate, per aumentarne la leggibilità faremo uso della macro `\num` del pacchetto `siunitx`.

Il seguente codice è l'intero contenuto del sub-template. Occorre un ciclo sulla variabile `sums` per produrre tutte le righe della rilevazione complessiva dei singoli reparti.

```

\begin{center}
\begin{tabular}{lcr}
\hline
\multicolumn{3}{c}{%
Inventario al 31/12/<<| year |>>}\}
\hline
\\
\multicolumn{3}{l}{%
Tabella riassuntiva per reparto}\}[2ex]
\hline
Nome reparto & Esistenza & Valore Esistenza
\euro\\
\hline
<<% for dep_name, dep_data
                        in sums.items() -%>>
Reparto <<| "%-18s"|format(dep_name)
|>> & \num{<<|
                        "%10d"|format(dep_data.qtytot)
|>>} & \num{<<|
                        "%0.2f"|format(dep_data.tottot)
|>>}\}
<<% endfor -%>>
\hline
Totale inventario & \num{<<|
                        "%10d"|format(sums.values()
                        |sum(attribute='qtytot'))
|>>} & \num{<<|

```

```

"%0.2f"|format(sums.values()
                |sum(attribute='tottot'))
|>>}\}\hline
\end{tabular}
\end{center}

```

Osservando il codice dei sub-template si nota che alcune operazioni di calcolo sono effettuate dal programma mentre altre dal template. Il compito di trovare il migliore equilibrio tra un template completamente privo di logica e uno che effettua tutte le elaborazioni sull'insieme minimo dei dati è lasciato al lettore come ulteriore esercizio. L'equilibrio intermedio dei template qui proposti è stato guidato da semplici considerazioni di efficienza nei calcoli dimostrando forse che le due situazioni estreme peccano la prima di complessità del codice e la seconda di efficienza del template.

Proprio le ultime righe del sub-template per generare il quadro riassuntivo provano come le somme parziali per fornitore e reparto sono lasciate allo script mentre i totali d'inventario sono lasciati al template attraverso la funzione filtro `sum()` che opera sulla lista dei valori del dizionario ottenuta con la funzione `values()`.

11.6 Il database

Ipotizziamo che il database esponga una vista che semplifichi la query degli articoli di magazzino. Una *vista* appare come una normale tabella di sola lettura del database mentre invece è ricavata al momento da una query prestabilita salvata permanentemente nella base di dati. A noi permetterà di non entrare nei dettagli della struttura dei dati, probabilmente articolata in ordini, fornitori, reparti, stagioni, articoli, rilevazioni d'inventario, eccetera. Del resto, la vista serve proprio a questo: costituire l'interfaccia ad alto livello ai dati nascondendone i dettagli. Se cambiasse la struttura interna la vista continuerebbe a essere la stessa una volta modificata adeguatamente la query sottostante.

11.7 Lo script

L'iterazione sulle tuple restituite dalla query crea il dizionario che memorizza l'intero insieme dei dati d'inventario chiamato `inv_data`. La sua struttura è rappresentata schematicamente nella figura 8 e corrisponde esattamente a quella attesa dal template. Completata la fase di reperimento delle informazioni dal database la funzione `calc_sums()` visita la struttura dati d'inventario per generare un secondo dizionario contenente i subtotali per fornitore e reparto.

Il codice del template principale è contenuto nel file `inventory.tex` nella sottocartella `templates`. Daremo quest'informazione all'oggetto `Environment` (tramite il `loader` di `Jinja`) in aggiunta alle stringhe utilizzate come delimitatori per le istruzioni e i campi espressione. Chiamando poi il template principale tramite il metodo `get_template()`, implicitamente verranno an-

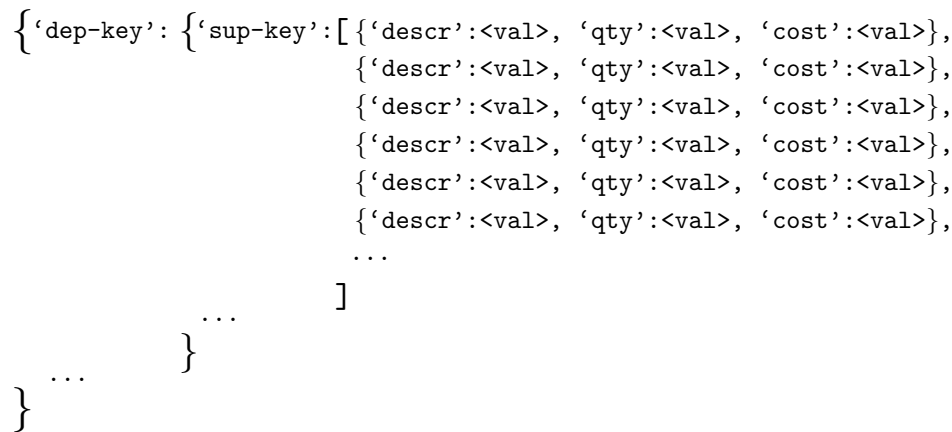


FIGURA 8: Schema della struttura a più livelli in cui sono memorizzati tutti gli articoli d'inventario. I simboli sono gli stessi dei tipi in Python ovvero le parentesi graffe definiscono un dizionario e le parentesi quadre una lista. I tre puntini indicano la ripetibilità dell'elemento dello stesso livello. Nel template questo dizionario è facilmente iterato su tre livelli di annidamento per generare il codice L^AT_EX di ambienti *tabular* multi-pagina dell'elenco degli articoli in inventario. Una struttura analoga è utilizzata nel codice applicativo per memorizzare i valori dei subtotali per fornitore e reparto. L'utente può disegnare schemi simili per facilitarsi la scrittura del codice.

che caricati i sub-template posizionati nella stessa directory.

Il codice si conclude chiamando il metodo `render()` dell'oggetto `template` e salvando il risultato testuale su disco pronto per essere compilato dal motore di composizione:

```
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-

from jinja2 import Environment
from jinja2 import FileSystemLoader
import sqlite3

year = 2014

# database section
conn = sqlite3.connect("../db/archive.sqlite")

# a dictionary for suppliers info
sups = dict() # key code => long name
for row in conn.execute("SELECT * FROM suppliers;"):
    sups[row[0]] = row[1]

# a dictionary with all warehouse items
inv_data = dict()

# the 'inv' table is a view
# data are ordered by code
query = """
SELECT code, descr, qty, cost, sup, dep
FROM inv
ORDER BY code;
"""
for item in conn.execute(query):
    dep = item[5]
    if not dep in inv_data:
        inv_data[dep] = dict()
    sup = item[4]

    if not sup in inv_data[dep]:
```

```
        inv_data[dep][sup] = []
        inv_data[dep][sup].append(
            {"descr":item[1],
             "qty":item[2],
             "cost":item[3]})

conn.close()

# utility template functions

# return supplier long name by code
def get_sup_name(sup):
    return sups[sup]

# a function to calculate the sums
def calc_sums(data):
    res = dict()
    for dep_name, sup_dict in data.items():
        if not dep_name in res:
            res[dep_name] = {'suptot':{}}
            qtytot = 0
            tottot = 0.0

            for sup_code, item_list
                in sup_dict.items():
                    if not sup_code in res[dep_name]:
                        res[dep_name][sup_code] = {}
                        qty = 0.0
                        tot = 0.0
                        for a in item_list:
                            qty += a['qty']
                            tot += a['qty']*a['cost']
                        res[dep_name]
                            ['suptot']
                            [sup_code] = {"qty":qty,
                                           "tot":tot}

                        qtytot += qty
                        tottot += tot

    res[dep_name]['qtytot'] = qtytot
    res[dep_name]['tottot'] = tottot
```

```

    return res

# template section
env = Environment(
    loader=FileSystemLoader('./templates'),
    variable_start_string="<<|",
    variable_end_string="|>>",
    block_start_string="<<%",
    block_end_string="%>>",
)

tmpl = env.get_template("inventory.tex")
res = tmpl.render({
    "year": year,
    "data": inv_data,
    "sums": calc_sums(inv_data)
})

# saving the content in a TeX file
f = open("inv-{0:d}.tex".format(year), "w")
f.write(res)
f.close()

```

12 Conclusioni

I problemi di elaborazione dati per la produzione di documentazione sono molto frequenti in ogni ambito e non è difficile acquisire le conoscenze per risolverli sviluppando script o programmi che producano in uscita i sorgenti per la composizione T_EX. Il vantaggio principale è che, tenendo separata la parte di elaborazione dati dalla parte della composizione tipografica, possiamo far evolvere e/o correggere singolarmente le due parti senza per questo dover stravolgere l'intero processo. L'elaborazione dei dati sarà influenzata dalla, e influenzerà la, scelta del linguaggio di programmazione in base alle strutture dati fornite e alle librerie disponibili, specialmente per il *templating*.

Nell'articolo sono stati presentati in dettaglio alcuni esempi di programmi scritti in bash, C, Lua e Python, privilegiando gli ultimi due linguaggi in ragione della loro diffusione e della loro semplicità rispetto al C. Si spera che la lettura di questo articolo aiuti l'utente a valutare se e come modificare l'approccio ai propri problemi di produzione della documentazione ogni volta che i dati abbiano un grado di complessità significativo.

L'ostacolo forse più grande per l'utente T_EX all'approccio della generazione automatica di sorgenti è acquisire i concetti di base del linguaggio di programmazione scelto. Esiste però almeno un metodo quasi alternativo: utilizzare LuaT_EX per elaborare i dati direttamente nel motore tipografico per poi comporre il documento con le tecnologie di interazione Lua-T_EX. Nemmeno così l'utente è esentato dall'imparare un linguaggio di programmazione — in questo caso obbligatoriamente Lua — per scrivere le funzioni di accesso ai database o tradurre i dati dai fogli elettronici, tuttavia nuove librerie di reporting in Lua per LuaT_EX potrebbero facilitare molto il compito di sviluppo.

Impostare l'intero processo in LuaT_EX è un metodo ancora tutto da valutare ma potenzialmente interessante perché la connessione con T_EX può essere molto più stretta di quanto non accada con la produzione separata dei sorgenti. Si spera che in futuro anche questo metodo si aggiunga pienamente alle soluzioni già disponibili e qui sinteticamente esposte.

13 Ringraziamenti

Ringraziamo l'amico Francesco Endrici, membro attivo della comunità del G_UU, per averci fornito un problema reale molto articolato di produzione di un documento a partire da un insieme di dati, perfetto per mostrare l'uso delle tabelle di Lua per descrivere informazioni strutturate.

Ringraziamo infine la Redazione di ArsTeXnica per tutto il prezioso lavoro fatto su questo articolo, dalla valutazione e revisione della bozza fino alla compilazione finale.

Riferimenti bibliografici

- ABS guide (2015). «Advanced bash-scripting guide». Web site <http://www.tldp.org/LDP/abs/html/index.html>.
- BATTISTIN, R. (2014). «L^AT_EX per la stesura dei rapporti di prova». *ArsTeXnica*, (18), pp. 55–63. URL <http://www.guitex.org/home/numero-18>.
- DE MARCO, A. e CRESCI, P. E. (2011). «L^AT_EX nella pubblica amministrazione. La web application FACILE per la produzione automatica di comunicazioni interne del Comune di Napoli». *ArsTeXnica*, (12), pp. 39–56. URL <http://www.guitex.org/home/it/numero-12>.
- DOWNEY, A. B. (2012). *Think Python: How To Think Like a Computer Scientist*. O'Reilly Media. URL <http://faculty.stedwards.edu/mikek/python/thinkpython.pdf>.
- FIANDRINO, C. (2014). «MenuT_EX: una piattaforma per realizzare menù». *ArsTeXnica*, (18), pp. 145–151. URL <http://www.guitex.org/home/numero-18>.
- GNU Bash (2015). «Bourne-again shell manual». <http://www.gnu.org/software/bash/manual/>.
- IERUSALIMSKY, R. (2013). *Programming in Lua, Third Edition*. Lua.Org.
- Jinja (2015). «Jinja is a full featured template engine for python». Official web site <http://jinja.pocoo.org/>.
- KERNIGHAN, B. W. e RITCHIE, D. M. (2004). *Il linguaggio C*. Pearson.

- KLEMENS, B. (2014). *21st century C*. O'Reilly and Associates.
- MUSTACHE (2015). «`{{mustache}}` logic-less templates». Official web site <https://mustache.github.io/>.
- OLIVINE-LABS (2015). «lustache - logic-less `{{mustache}}` templates with lua». Official web site <http://olivinelabs.com/lustache/>.
- PILGRIM, M. (2009). *Dive Into Python 3*. Apress. URL <http://getpython3.com/diveintopython3/>.
- SQLite3 (2015). «The sqlite consortium». Web site <https://www.sqlite.org/>.
- TANTAU, T. (2015). «PGF manual version 3.0.1». Available on www.ctan.org.
- ▷ Roberto Giacomelli
giaconet dot mailbox at gmail dot com
 - ▷ Gianluca Pignalberi
g dot pignalberi at alice dot it

Ricordi di un cacciatore di spazi spuri

Enrico Gregorio

Sommario

Parecchi trabocchetti ci attendono quando programiamo in $(\LaTeX)$. Esamineremo alcuni di essi in cerca di una soluzione. Troveremo la terra promessa? Forse, con `expl3`.

Abstract

Several pitfalls are waiting for us when programming in $(\LaTeX)$. This paper will examine some and search for a solution. Shall we find the promised land? Maybe so, with `expl3`.

1 Introduzione

Di per sé la programmazione in $(\LaTeX)$ non è facile, principalmente per le idiosincrasie del linguaggio, ma anche per via di alcune sottigliezze dovute al fatto che il linguaggio è orientato alla *composizione tipografica* del testo. Knuth ha fatto del suo meglio per facilitare la scrittura di un documento senza preoccuparsi troppo degli spazi: in circostanze normali serie di spazi o di tabulazioni sono considerate come uno spazio singolo; la fine riga è convertita in uno spazio, purché non sia seguita da un'altra fine riga separata dalla precedente da soli spazi; lo spazio bianco all'inizio della riga è ignorato; gli spazi sono rimossi dalla fine di una riga e sostituiti da uno speciale marcatore.

Le regole precise sono spiegate nei dettagli nel *TeXbook* (KNUTH, 1986) e in *TeX by Topic* (EIJKHOUT, 1992), questo articolo non è il posto dove entrare nelle minuzie. Ritornerò su alcuni dei punti descritti prima.

Alcuni dei lettori potrebbero conoscermi per la mia attività su TeX.StackExchange (come utente **egreg**) dove, secondo il parere di uno dei più stimati membri della comunità, ho accumulato gran parte della mia reputazione scovando spazi spuri in codice $\TeX$.¹ A dire il vero, scovare spazi spuri è talvolta piuttosto complicato: possono nascondersi in angoli bui e può essere necessario ricorrere a `\tracingall` o altre armi pesanti per isolarli: le macro di $\TeX$ si richiamano a vicenda in modi talvolta molto contorti.

Nessuno che abbia intrapreso il compito di scrivere macro, semplici o anche terribilmente complicate, è mai stato immune dal morso di qualche spazio spurio lasciato nel codice. Succede! Una delle ragioni più frequenti è la risistemazione del codice:

1. Ci piace scherzare nella *chatroom* del sito; non sono però del tutto sicuro che quella osservazione sia uno scherzo.

una riga è troppo lunga e vorremmo migliorare la leggibilità del codice, così la spezziamo e dimentichiamo il magico `%` a fine riga. Un altro aspetto importante del linguaggio di $\TeX$ è che in certe situazioni gli spazi sono *ignorati*: ci sono e $\TeX$ si comporta come se non ci fossero, ma eseguendo un'azione particolare che descriverò più avanti.

Dobbiamo distinguere con attenzione tra *battute spazio* e *token spazio*: solo le *battute spazio* (quando si spinge il pollice sulla barra spaziatrice scrivendo il documento) sono soggette alla regola di contrazione descritta prima, mentre i *token spazio* non lo sono. Un semplice esempio: `\space\space` produrrà *due spazi*, perché quando $\TeX$ espande `\space`, lo converte in un token spazio.

Questo articolo descriverà gli errori più comuni, con esempi presi da domande su TeX.StackExchange o da codice di pacchetti. Non darò riferimenti precisi, perché lo scopo non è di svergognarne gli autori.² Finirò con qualche considerazione sui metodi per evitare questi problemini.

L'articolo è una traduzione di "Recollections of a spurious space catcher" che apparirà negli atti del TUG meeting 2015, *TUGboat* 35 (2015).

2 Primi esempi

Un famoso *spaghetti western* è *Il buono, il brutto e il cattivo* di Sergio Leone, con Clint Eastwood, Lee Van Cleef e Eli Wallach. Mi piace presentare gli esempi in questa forma e così farò, per cominciare.

2.1 Il brutto

```
\providecommand{\sVert}[1][0]{
\ensuremath{\mathinner{
\ifthenelse{\equal{#1}{0}}{ % if
\rvert}{ }
\ifthenelse{\equal{#1}{1}}{ % if
\big\rvert}{ }
\ifthenelse{\equal{#1}{2}}{ % if
\Big\rvert}{ }
\ifthenelse{\equal{#1}{3}}{ % if
\biggr\rvert}{ }
\ifthenelse{\equal{#1}{4}}{ % if
\Biggr\rvert}{ }
}} % \ensuremath{\mathinner{
}
```

Questo codice fa parte di un pacchetto nel quale le definizioni non sono separate da righe vuote e che è brutto sotto molti aspetti: non c'è alcun rientro che possa chiarire le relazioni fra le varie

2. Magari farò un paio di eccezioni.

```
\cref@addlanguage@def{spanish}{%
\PackageInfo{cleveref}{loaded 'spanish' language definitions}
\renewcommand{\crefrangeconjunction}{\nobreakspace}%
\renewcommand{\crefrangepreconjunction}{}%
\renewcommand{\crefrangepostconjunction}{}%
\renewcommand{\crefpairconjunction}{\nobreakspace}%
[...]
```

TABELLA 1: Codice da una vecchia versione di `cleveref.sty`

parti e l'annidamento dei condizionali; il codice è goffo e trascura molte protezioni a fine riga.

Ricordo che una fine riga è convertito in un token spazio; in questo esempio particolare non sono particolarmente importanti, perché la macro deve essere usata in modo matematico, nel quale i token spazio sono ignorati. Ma un utente *potrebbe* adoperare `\sVert` in modo testo per via di `\ensuremath`, finendo con spazi in eccesso: in questo caso la prima fine riga e lo spazio tra `}}` e `%` alla fine *non* sono ignorati.

L'uso di `\providecommand` è ovviamente sbagliato: un utente che carichi il pacchetto si aspetta che il comando `\sVert` esegua l'azione descritta, non qualche altra. Una definizione migliore può essere

```
\newcommand*{\sVert}[1][0]{%
\ifcase#1\relax
\rvrt % 0
\or
\big\rvrt % 1
\or
\Big\rvrt % 2
\or
\biggr\rvrt % 3
\or
\Biggr\rvrt % 4
\fi
}
```

Considero cattivo stile di programmazione avere `%` accanto a una parola di controllo, ma è solo la mia opinione. I caratteri `%` dopo `\rvrt` non sarebbero necessari se non ci fossero i commenti.

2.2 Il cattivo

```
\def\@wrqbar#1{%
\ifnum\value{page}<10\def\X{\string\X}\else%
\ifnum\value{page}<100\def\X{\string\Y}\else%
\def\X{\string\Z}\fi\fi%
\def\F{\string\F}\def\E{\string\E}%
\stepcounter{arts}%
\iffootnote%
\edef\@tempa{\write\@barfile{\string%
\quellentry{#1\X}{\thepage}}{\F}{\thefootnote}}}%
\else%
\edef\@tempa{\write\@barfile{\string%
\quellentry{#1\X}{\thepage}}{\E}{\thearts}}}%
\fi%
\expandafter\endgroup\@tempa%
\if@nbreak \ifvmode\nobreak\fi\fi\@esphack}
```

Qui di sicuro non troveremo spazi spuri! Ci sono perfino troppi caratteri `%`! Il problema è

che è impossibile leggere il codice, tanto meno comprenderlo.

2.3 Il buono

```
\def\deleterightmost#1{%
\edef#1{\expandafter\xyzy#1\xyzy}}
\long\def\xyzy\#1#2{\ifx#2\xyzy\yzzyx
\else\noexpand\{#1}\fi\xyzy#2}
\long\def\yzzyx#1\xyzy\xyzy{\fi}
```

Questo codice del Grand Wizard può essere preso come modello di chiarezza. Non è altrettanto chiaro che cosa faccia: per capirlo è necessario leggersi l'appendice D del *T_EXbook*.

2.4 Sorpresa!

```
{\tt A'C B}

\def\adef#1{\catcode'#1=13 \begingroup
\lccode'\~='#1\lowercase{\endgroup\def~}}
\let\oldtt\tt\def\tt{\adef'\char"0D}\oldtt}

{\tt A'C B}

{\tt A'c B}
```

Fate andare plain T_EX su questo codice (aggiungendo `\bye` alla fine) e ne riceverete una bella sorpresa. Lo scopo è di sostituire l'apostrofo curvo con quello diritto che il font `cmtt10` ha alla posizione "0D. La sorpresa è che viene stampato solo AB senza alcuno spazio tra le lettere, né apostrofo, né C. Però troviamo un avviso misterioso nel log:

```
Missing character:
There is no ^dc in font cmtt10!
```

Questo esempio non ha spazi in più, invece ne *manca* uno!

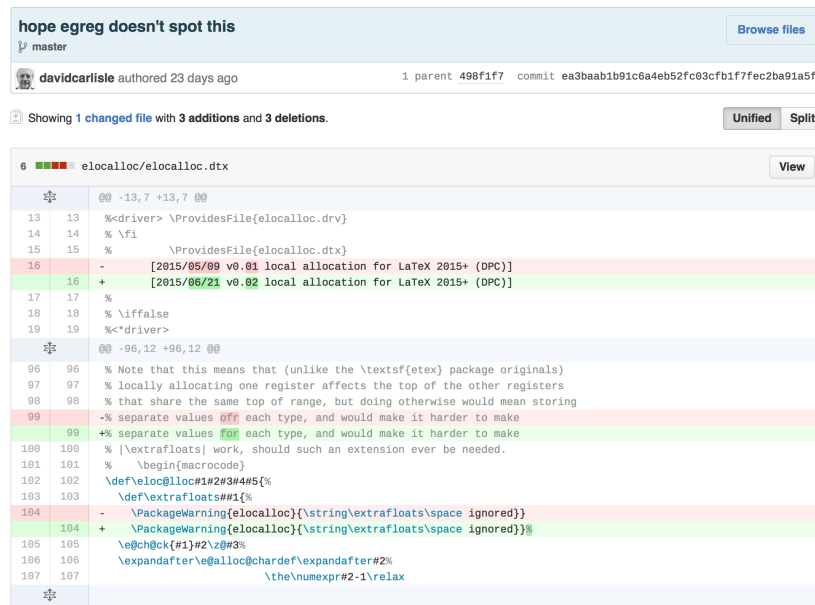
2.5 I grandi programmatori non sono immuni

La tabella 1 mostra un estratto da `cleveref.sty`, che è certamente un software notevole.

Forse l'autore aveva provato il supporto per le lingue solo in alcuni casi: lo spazio spurio si notava solo quando un utente provava qualcosa come

```
The Spanish word for Spain is
\foreignlanguage{spanish}{Espana~na}
```

e riscontrava la presenza di *due spazi* tra 'is' e 'España'. Isolare il problema non fu affatto facile,

FIGURA 1: Un altro esempio che anche grandi T_EXperts non sono immuni

perché `cleveref` non sembrava nemmeno coinvolto. La macro `\cref@addlanguagedefs` aggiunge codice a `\extrasspanish`, che viene eseguita ogni volta che la lingua passa allo spagnolo: a ogni cambio di lingua veniva prodotto uno spazio.

Si veda la figura 1 per un altro esempio.

3 Che succede?

La regola è semplice: una fine riga è convertita in una *battuta spazio* che può diventare un token spazio, ma non se segue una parola di controllo. Quindi, nella riga

```
\iffootnote%
```

il carattere di commento non è necessario perché la battuta spazio che risulta dalla fine riga sarà ignorata, proprio perché segue una parola di controllo. Al contrario la fine riga in

```
\PackageInfo{cleveref}{loaded 'spanish'
  language definitions}
```

diventa un normale token spazio nel testo di sostituzione della macro che si sta definendo e sparirà solo se T_EX è di buon umore quando la macro viene usata.

T_EX è di buon umore nei confronti dei token spazio quando il modo corrente è verticale (tra capoversi, essenzialmente) o matematico: in questi casi i token spazio non fanno nulla. Se T_EX sta componendo un normale capoverso (o una box orizzontale), i token spazio non sono di solito ignorati e questa è la causa del misterioso effetto presentato nella sezione 2.5.

Ecco alcuni esempi di codice scritto da persone probabilmente abituate ad altri linguaggi di programmazione, in cui gli spazi sono usati molto più

liberamente per separare i token e sono per lo più irrilevanti (purché non in una stringa).

```
\newcommand{\smallx}[1]{
  \begin{center}
    \begin{Verbatim}[commandchars=\\\{\}]

      \code{#1}

    \end{Verbatim}
  \end{center}
}
=====
\include{fp}
\newcommand\entryOne[1]{
  \ifnum #1 = 100 #1 \fi
}
\newcommand\entryTwo[1]{
  \FPeval{\result}{#1}
  \ifnum \result = 100 \result \fi
}
=====
\newcommand{\trellis}[4]{
  \def \STATES {#1}
  \def \PSK {#2}
  \def \XDISTANCE {#3}
  \def \YDISTANCE {#4}
  \FPupn\NGROUPS{\STATES}{ \PSK}{ div 0 trunc}
  \multido{\ryA=0+-\YDISTANCE,\nA=1+1}{\STATES}{%
    \dotnode(0,\ryA){dotA\nA}
    \dotnode(\XDISTANCE,\ryA){dotB\nA}
  }
  \multido{\nG=1+1,\nOffset=1+\PSK}{\NGROUPS}{%
    \multido{\nStart=\nG+\NGROUPS}{\PSK}{%
      \multido{\nArrows=\nOffset+1}{\PSK}{%
        \ncline{dotA\nStart}{dotB\nArrows}
      }
    }
  }
}
```

Gli autori dei primi due esempi non sanno nulla di spazi spuri e scrivono il codice T_EX come se fosse C. Il terzo esempio mescola le protezioni a

SENATVSPOPVLVSQVEROMANVS
IMPCAESARIDIVINERVAEFNERVAE
TRAIANOAVGGERMDACICOPONTIF
MAXIMOTRIBPOTXVIIIMPVICOSVIPP
ADDECLARANDVMQVANTAEALTITVDINIS
MONSETLOCVSTANTISOPERIBVSSITEGESTVS

FIGURA 2: L'iscrizione sul basamento della *Columna Traiana* a Roma

fine riga con *free form code*. Contare il numero degli spazi spuri è divertente.

Sfortunatamente TEX non è *free form*! Gli spazi sono importanti nella composizione, a meno che non vogliamo scrivere i nostri testi come facevano gli antichi romani: nella figura 2 possiamo vedere un'emulazione dell'iscrizione sul basamento della *Columna Traiana* a Roma.³ Il font è Trajan di Peter Wilson.

Gli antichi avevano buone ragioni per non usare spazi: i suffissi facilitavano la divisione in parole, ma forse più importante era risparmiare spazio: il marmo e la pergamena erano molto costosi. Solo l'introduzione della stampa e di carta più economica permise di usare spazi tra le parole per maggiore chiarezza e facilità di lettura.

Bene, uno potrebbe dire, perché non mettiamo % alla fine di ogni riga nel codice di programmazione e smettiamo di preoccuparci, anche se il codice sarà un po' più arduo da leggere?

Spiacente, non funziona. Ecco un altro esempio che lo dimostra: ci sono due malattie che chiamo 'sindrome da spazio spurio' e 'sindrome da spazio mancante'. La seconda è più grave.

```
\documentclass{article}
\newcount\monthlycount
\newcommand{\monthlytodo}[1]{\par%
  \fbox{%
    \parbox{10cm}{%
      \monthlycount=1%
      \loop\ifnum\monthlycount<13%
        #1--\number\monthlycount\hrulefill\par%
        \advance\monthlycount by 1%
      \repeat%
    }%
  }%
}
\begin{document}
\monthlytodo{2013}
\end{document}
```

Questo codice dovrebbe servire a stampare una lista numerata e incorniciata, con gli elementi della lista preceduti da anno e numero del mese. Se proviamo a compilarlo, dopo un bel po' di tempo senza segni di attività, TEX si fermerà con il messaggio

3. Mentre scrivevo l'articolo, sono stato vittima di uno spazio spurio che si è inserito nel testo dell'iscrizione, ma fortunatamente l'ho notato prima di inviarlo.

```
! TeX capacity exceeded, sorry
[main memory size=5000000].
<to be read again>
```

1.15 \monthlytodo{2013}

Sorprendente, no? L'autore del codice era stato, in precedenza, vittima della sindrome da spazio spurio, quindi cominciò ad aggiungere % alla fine di ogni riga, perfino nel documento stesso, non solo nel codice del preambolo.

Vediamo che succede: il testo di sostituzione della macro verrebbe mostrato da TEX come

```
... \loop \ifnum \monthlycount
  <13#1--\number \monthlycount ...
```

e, quando \monthlytodo{2013} è chiamato, il test diventa

```
... \loop \ifnum \monthlycount
  <132013--\number \monthlycount ...
```

Non ci meravigliamo più che TEX ci metta tanto a finire il ciclo e che esaurisca la memoria, perché sta tentando di costruire una \fbox.

Un caso conclamato di sindrome da spazio mancante. La soluzione è di *non* avere % dopo 13. Questa costante è parte di un test numerico e quindi siamo in uno dei casi speciali in cui TEX *ignora* un token spazio che la segua.

Il codice della sezione 2.4 soffre della stessa sindrome; quando {\tt A'C B} è sviluppato, il carattere apostrofo diventa attivo ed espanso come una macro, producendo la lista di token

```
A\char"0DC B
```

ed è qui che va tutto a pallino: \char si aspetta un numero in formato esadecimale per via di " e trova le cifre DC; dal momento che in quella posizione del font non c'è alcun carattere, compare l'avviso di Missing character ~^dc|. Il codice corretto deve avere uno spazio oppure \relax:

```
\let\olddtt\tt
\def\tt{\adef'\{\char"0D }\olddtt}

\let\olddtt\tt
\def\tt{\adef'\{\char"0D\relax}\olddtt}
```

Quale dei due è una decisione stilistica: sia lo spazio sia \relax terminano la ricerca di altre cifre; \relax non fa nulla mentre lo spazio è ignorato. C'è un'altra possibilità:

```
\let\olddtt\tt\def\tt{\adef'\{active}\olddtt}
```

ma qui si adopera il fatto accidentale che "0D = 13 and che \active è definita con \chardef per puntare al carattere 13: non un esempio di buona programmazione.

Un esempio più perfido, scoperto da Frank Mittelbach qualche giorno dopo il convegno del TUG.⁴

4. <http://tex.stackexchange.com/questions/257100/varioref-and-previous-page>

```
2337 \advance\@tempcnta-2%
2338 \ifnum \thevpagerefnum =\@tempcnta%
```

Ricordate lo spazio spurio in `cleveref.sty` discusso nella sezione 2.5? Dopo che segnalai il baco, l'autore aggiunse % alla fine di ogni riga. Ma il risultato è un grave caso di sindrome da spazio mancante: quando $\TeX$ si aspetta una costante numerica, guarda se trova un token spazio o un token non espandibile che non possa essere interpretato come cifra. Per questa ricerca, i token vengono *espansi*, in particolare il condizionale è valutato prima che un valore sia assegnato alla variabile `\@tempcnta` perché $\TeX$ ancora non sa se il numero è terminato. Il risultato è che il riferimento non sarà corretto. Il codice è parte della ridefinizione di un comando in `varioref.sty`; il codice originale (quasi identico) non ha % dopo -2.

La regola precisa è che quando un costrutto sintattico permette *one optional space*, $\TeX$ lo cercherà eseguendo l'espansione. Cito il *TeXbook* (KNUTH, 1986, p. 208)

Per un risultato ottimale, mettete sempre uno spazio bianco dopo una costante numerica; questo spazio bianco dice a $\TeX$ che la costante è completa e questo spazio non apparirà in stampa. Di fatto, se non mettete lo spazio bianco dopo una costante, $\TeX$ deve fare più lavoro, perché una costante continua fino a che una non-cifra viene letta; se questa non-cifra non è uno spazio, $\TeX$ la rimette al suo posto, pronta per essere letta di nuovo.

Si potrebbe obiettare che chi usa `\loop` dovrebbe conoscere i dettagli della programmazione $\TeX$, dal momento che questa macro non è ufficialmente supportata da $\LaTeX$ e quindi il problema dell'esaurimento della memoria non si presenterebbe. Una stima grossolana del numero di pacchetti che usano `\loop` è 378. Non ho nemmeno provato a controllare quanti usino `\ifnum`: entrambi sono essenziali nella cassetta degli attrezzi di qualsiasi programmatore ($\LaTeX$).

È evidente che un aspirante autore di macro dovrebbe essere a conoscenza di questi problemi, ma sono molto sottili e, come abbiamo visto, anche programmatori molto esperti possono cadere vittima della grave sindrome. Potrei dare molti altri esempi.

4 Soluzioni

Una via d'uscita dal problema delle sindromi collegate agli spazi potrebbe essere quella di delimitare la parte di programmazione con qualcosa che sia equivalente a

```
\catcode' =9 \endlinechar=-1 \makeatletter
```

(l'ultimo comando è per maneggiare le macro interne di $\LaTeX$) con un codice di riallineamento alla fine, qualcosa come

```
\edef\restorecodes{%
  \catcode32=\the\catcode32
  \endlinechar=\the\endlinechar
  \noexpand\makeatother
}
\catcode32=9 \endlinechar=-1 \makeatletter

<macro code>

\restorecodes
```

ma questo non curerebbe la sindrome da spazio mancante; anzi, la aggraverebbe perché non potremmo più aggiungere uno spazio alla fine di una costante! Ancora. Talvolta i token spazio servono nel codice di una macro. Potremmo scrivere `\space` dove ci serve uno spazio, ma sarebbe goffo. Ci viene in mente allora una brillante idea: usare ~ come spazio. Proviamo.

```
\edef\restorecodes{%
  \catcode32=\the\catcode32
  \catcode126=\the\catcode126
  \endlinechar=\the\endlinechar
  \noexpand\makeatother
}
\catcode32=9 \catcode126=10
\endlinechar=-1 \makeatletter

\newcount\monthlycount
\newcommand{\monthlytodo}[1]{\par
  \fbox{
    \parbox{10cm}{
      \monthlycount=1~
      \loop\ifnum\monthlycount<13~
        #1--\number\monthlycount\hrulefill\par
        \advance\monthlycount by 1~
      \repeat
    }
  }
}

\restorecodes
```

Molto più *free form*. Ricordo che il codice di categoria 9 significa 'ignorato' e 10 significa 'spazio'. Un momento! La regola dice che gli spazi vengano rimossi alla fine di una riga e sostituiti con il marcatore interno, in questo caso nulla perché il parametro `\endlinechar` ha il valore -1. E forse non avremmo 'veri spazi' nel testo di sostituzione delle nostre macro. Un paio di citazioni dal *TeXbook* ci tolgono i dubbi. La prima riguarda la rimozione degli spazi (KNUTH, 1986, p. 46)

$\TeX$ cancella ogni carattere *spazio* (numero 32) che capitino alla fine di una riga di input. Poi inserisce un carattere *return* (numero 13) alla fine della riga [...]

A questo va aggiunto il fatto che `\endlinechar` di solito ha il valore 13. Quindi la tilde con codice

di categoria 10 non viene rimosso perché non ha codice carattere 32.⁵

Per il secondo problema, vediamo (KNUTH, 1986, p. 47)

Se T_EX vede un carattere di categoria 10 (spazio), l'azione dipende dallo stato corrente. Se T_EX è in stato *N* o *S*, il carattere è semplicemente saltato e T_EX rimane nello stesso stato. Altrimenti T_EX è in stato *M*; il carattere è convertito in un token di categoria 10 e con codice di carattere 32, dopo di che T_EX va in stato *S*. Il codice di carattere in un token spazio è sempre 32.

Possiamo dunque contare sul fatto che ~ venga convertito in un 'vero token spazio' quando il testo di sostituzione di una macro è messo in memoria. Certo, perdiamo il carattere che indica uno spazio senza possibilità di spezzare una riga, ma questo ci serve raramente, scrivendo macro, e possiamo sempre adoperare `\nobreakspace`.

Perché non usare `\relax`, invece? Può certamente essere usato, ma perderemmo l'espandibilità 'completa' che ci serve in molte situazioni. Notiamo anche che i registri numerici e le costanti definite con `\chardef` o `\mathchardef` sono già numeri 'completi' e non c'è la ricerca di spazi dopo di essi.

5 L^AT_EX₃ e exp13, un nuovo modo per evitare questi problemi, ma anche molto di più

Chiunque abbia la gentilezza di seguire le mie risposte su TeX.StackExchange avrà già capito dove mi sto dirigendo: il progetto L^AT_EX₃ fu avviato più di due decenni fa, ma è rimasto inerte a lungo, fino a pochi anni fa, quando divenne effettivamente possibile usare il nuovo paradigma di programmazione che aveva introdotto. Quando il L^AT_EX team cominciò a studiarlo, le risorse di calcolo erano davvero limitate: per esempio, L^AT_EX 2_ε su emT_EX lasciava pochissimo spazio per etichette e macro personali. Al giorno d'oggi, invece, possiamo produrre presentazioni, diagrammi e disegni complicati con tempi di calcolo molto brevi. Ricordo senza alcuna nostalgia i tempi in cui anche un diagramma non troppo complicato in P_TC_TE_X richiedeva parecchi minuti! Produrre una presentazione con 37 schermate da questo articolo richiede pochi secondi, nonostante la doppia compilazione con un processo PythonT_EX in mezzo.

Il rallentamento dovuto a dover leggere qualche migliaio di righe di codice non è più un problema come anni fa; quando queste righe di codice verranno incorporate nel formato, il tempo di caricamento diventerà di pochi millisecondi.

5. Occorre menzionare che le attuali implementazioni di T_EX in T_EX Live e MiK_TE_X rimuovono anche i tabulatori (codice di carattere 9).

L'ambiente di programmazione exp13 fornisce delimitatori dei blocchi di codice simili a quelli che ho descritto prima, ma tratta anche _ e : come caratteri che possono essere usati nei nomi delle sequenze di controllo. Si può pensare a _ come al tradizionale @ (che invece in quei nomi non è lecito); l'uso dei due punti è interessante, ci tornerò dopo aver presentato alcuni esempi.

Sono ben conscio del fatto che cambiare un paradigma di programmazione può essere difficile all'inizio, perché è duro liberarsi delle abitudini. Un paio di problemini di assaggio potrebbero essere utili per attirare l'attenzione.

Primo problema: desideriamo ottenere il rapporto tra due lunghezze in modo espandibile da usare, per esempio, come fattore moltiplicativo di un'altra lunghezza, e vorremmo che il rapporto fosse espresso con la migliore approssimazione possibile. Il codice si vede nella tabella 2.

Il codice è su più righe per maggiore leggibilità, ma la parte importante è essenzialmente una riga di codice! E può perfino essere *free form*! Si adopera la funzione `\fp_eval:n` che produce il risultato del calcolo, insieme a una funzione che può convertire da un tipo di dati a un altro. Se proviamo `\dimratio{\textwidth}{\textheight}` o `\dimratio[2]{\textwidth}{\textheight}` otteniamo, rispettivamente, 0.62727 e 0.63; potremmo anche dire

```
\setlength{\mylength}{%
  \dimratio{\textwidth}{\textheight}\mylength
}
```

per scalare del dato rapporto il valore del parametro `\mylength`.

Secondo problema. Vorremmo isolare e trattare gli elementi in una lista in cui il separatore è il punto e virgola. Prima un interessante codice di Petr Olšák:

```
\def\ls#1{\lsA#1;;}
\def\lsA#1{%
  \ifx;#1;%
    \else
      \dosomething{#1}\expandafter\lsA
    \fi
  }
\def\dosomething#1{%
  \message{I am doing something with #1}%
}

\ls{(a,b);(c,d);(e,f)}
```

Si tratta di una tecnica ben nota che fa bene il suo mestiere, ma ha alcuni difetti che illustrerò dopo aver mostrato il corrispondente codice exp13:

```
\Exp1SyntaxOn
\NewDocumentCommand{\dosomething}{m}
{
  I ~ am ~ doing ~ something ~ with ~ #1
}
```

```

\documentclass{article}
\usepackage{xparse}

\ExplSyntaxOn % start the programming environment
\DeclareExpandableDocumentCommand{\dimratio}{ 0{5} m m }
{
  \fp_eval:n
  {
    round ( \dim_to_fp:n { #2 } / \dim_to_fp:n { #3 } , #1 )
  }
}

\ExplSyntaxOff % end the programming environment

```

TABELLA 2: Codice per la macro `\dimratio`

```

\seq_new:N \l_manual_ls_items_seq

\NewDocumentCommand{\ls}{m}
{
  \seq_set_split:Nnn
  \l_manual_ls_items_seq
  { ; }
  { #1 }
  \seq_map_function:NN
  \l_manual_ls_items_seq
  \dosomething
}
\ExplSyntaxOff

\ls{(a,b); (c,d); (e,f)}

```

Uno dei nuovi tipi di dati introdotti da `expl3` è il tipo *sequence*: una lista ordinata di elementi ai quali si può accedere globalmente o per numero d'ordine. La prima funzione separa gli elementi e li ordina nella *sequence*; poi `\seq_map_function:NN` passa ogni elemento come argomento alla funzione `\dosomething`, esattamente lo stesso che fanno le macro di Petr Olšák.

Un momento! Se si guarda con attenzione, nel mio esempio ci sono spazi attorno all'elemento centrale che nel codice di Petr passerebbero come parte dell'argomento a `\dosomething`. Questo non succede nel codice `expl3`, perché la funzione di separazione automaticamente rimuove spazi che precedono o seguono un elemento. In particolare, si può perfino scrivere

```

\ls{
  (a,b);
  (c,d);
  (e,f)
}

```

se lo si ritiene conveniente.

Torniamo al codice del Grand Wizard mostrato nella sezione 2.3

```

\def\deleterightmost#1{%
  \edef#1{\expandafter\xyzy#1\xyzy}}

```

```

\long\def\xyzy\#1#2{\ifx#2\xyzy\yzzyx
  \else\noexpand\#1\fi\xyzy#2}
\long\def\yzzyx#1\xyzy\xyzy{\fi}

```

Lo scopo è di rimuovere l'ultimo elemento da una *sequence*. Nell'appendice D (KNUTH, 1986, p. 378) vengono chiamate *list macros* e sono implementate come macro con un testo di sostituzione come `\{(a,b)\{(c,d)\{(e,f)\}`; la definizione della *sequence* sarebbe

```
\def\myitems{\{(a,b)\{(c,d)\{(e,f)\}}
```

e la macro `\deleterightmost` deve essere chiamata come

```
\deleterightmost\myitems
```

per rimuovere l'ultimo elemento, come se la prima definizione fosse stata

```
\def\myitems{\{(a,b)\{(c,d)\}}
```

Bene, paragonate il codice di Knuth con quello `expl3`

```

\seq_pop_right:NN
  \l_manual_ls_items_seq
  \l_tmpa_tl

```

il quale ha anche il grande vantaggio che l'ultimo elemento è ancora disponibile nella variabile *token list* temporanea `\l_tmpa_tl` (o qualsiasi altra variabile dello stesso tipo decidiamo di usare). La macro di Knuth invece lo scarta.

Ecco una nuova versione della macro `\monthlytodo`:

```

\documentclass{article}
\usepackage{xparse}

\ExplSyntaxOn
\NewDocumentCommand{\monthlytodo}{ 0{10cm} m }
{
  \par
  \noindent
  \fbox
  {
    \manual_monthlytodo:nn { #1 } { #2 }
  }
}

```

```

}
\cs_new_protected:Nn \manual_monthlytodo:nn
{
  \parbox
  {
    \dim_eval:n { #1 - 2\fbboxsep - 2\fbboxrule }
  }
  {
    \int_step_inline:nnnn { 1 } { 1 } { 12 }
    {
      #2--\int_to_arabic:n { ##1 }
      \hrulefill
      \par
    }
  }
}
\ExplSyntaxOff

\begin{document}

\monthlytodo{2013}

\monthlytodo[\textwidth]{2015}

\end{document}

```

Notiamo che non c'è più `\loop`, ma si adopera la funzione `\int_step_inline:nnnn`, molto più maneggevole, che ha come argomenti il punto di partenza, il passo, il punto di arrivo e il codice da eseguire, nel quale il valore corrente è disponibile come `#1`, che, in questo caso, deve diventare `##1` perché stiamo dando una definizione. Un argomento facoltativo permette di decidere la larghezza della box: un espediente per dimostrare l'uso di `\dim_eval:n`.

Fine delle ossessioni sugli spazi dopo le costanti, perché gli oggetti sono chiaramente separati gli uni dagli altri. Se una funzione vuole un argomento numerico, questo sarà tra graffe. La *segnatura* della funzione (cioè la parte dopo i due punti) ci dice quanti argomenti sono richiesti (ma ovviamente bisogna anche sapere che tipo di argomento è atteso).

Mostriamo la potenza di `expl3` costruendo un calendario bimensile:

```

\cs_new_protected:Nn \manual_bimonthlytodo:nn
{
  \parbox
  {
    \dim_eval:n { #1 - 2\fbboxsep - 2\fbboxrule }
  }
  {
    \int_step_inline:nnnn { 1 } { 2 } { 12 }
    {
      #2 --
      (\int_to_arabic:n { ##1 } --
      \int_to_arabic:n { ##1 + 1 })
      \hrulefill\par
    }
  }
}

```

Qui il passo è due; verranno eseguiti solo sei passi, perché il successivo andrebbe oltre il limite superiore stabilito: lasciamo i calcoli a T_EX.

In `expl3` c'è un'attenta distinzione tra *funzioni* e *variabili*. Internamente sono ovviamente realizzate

con macro o registri, ma la distinzione è nell'uso: le funzioni eseguono qualcosa, le variabili contengono token o valori.

Lo schema di nomi rende facile la distinzione: una funzione ha una segnatura formata da zero o più caratteri, separati dal nome con i due punti. Il manuale introduttivo (THE L^AT_EX PROJECT, 2015b) spiega quali caratteri sono leciti e che cosa significano. Il nome di una variabile deve cominciare con `l_`, `g_` o `c_`, a significare *locale*, *globale* o *costante*; `expl3` fornisce funzioni distinte per agire sulle variabili locali o globali; le costanti vanno solo definite dando loro un valore. Un modo facile per trovarsi nelle peste è di esaurire la memoria perché si è riempito completamente il *save stack*; agire sempre correttamente sulle variabili evita il problema. Il manuale sull'interfaccia (THE L^AT_EX PROJECT, 2015a) descrive tutte le funzioni rese disponibili dal *kernel*; ci sono anche altri manuali per i pacchetti ancora allo stato sperimentale, per esempio quello sulle espressioni regolari (THE L^AT_EX PROJECT, 2015c). Non va trascurato il pacchetto per la definizione di comandi di alto livello *xparse* (THE L^AT_EX PROJECT, 2015d).

6 Vantaggi e svantaggi

Attrezzi potenti hanno sempre i loro pro e contro. Ecco una breve lista di ciò di buono che ho trovato in `expl3`.

1. Interfaccia coerente: il team si sforza di fornire un insieme di funzioni di base in modo che il nome suggerisca l'azione.
2. Nuovi tipi di dati: T_EX ha, essenzialmente, solo macro, ma `expl3` costruisce strutture di più alto livello che aiutano nel tenere separati i concetti.
3. Centinaia di funzioni predefinite: le azioni più comuni sulle varie strutture di dati sono coperte e il team è di solito pronto a rispondere quando nuovi casi d'uso vengono dimostrati.
4. Sviluppo in corso: lo stato di `expl3` è abbastanza stabile; il buono è che `expl3` nasconde i dettagli implementativi delle azioni di base e dei tipi di dati, in modo che cambi a basso livello non abbiano alcun effetto sui costrutti di più alto livello (efficienza e velocità a parte).

Una breve lista dei nuovi tipi di dati:

token list è un contenitore generico di token (si pensi a `\chaptername`);

sequence è un insieme ordinato di *token list*;

comma separated list è quasi lo stesso, ma più orientato verso l'interfaccia utente;

property list è un insieme disordinato di *token list*, con indirizzamento per *chiave*;

floating point number secondo lo standard IEEE;

regular expression è il più possibile vicino allo standard POSIX;

coffin è una normale box di T_EX, ma con molte più maniglie.

I tipi di dati comprendono anche *boolean*, *integer*, *box*, *length*, *skip* (cioè le lunghezze elastiche) oltre ai flussi di input e output come nello standard T_EX.

Un altro esempio. Molti articoli su TUGboat riguardano il problema di macro per eseguire il *case switching*; ecco come si può fare in `expl3`:

```
\str_case:nnTF { <stringa> }
{
  {a}{Caso-a}
  {b}{Caso-b}
  {z}{Caso-z}
}
{
  Codice aggiuntivo se
  c'è una corrispondenza
}
{
  Codice da eseguire se non
  c'è una corrispondenza
}
```

La *<stringa>* nel primo argomento proverrà di solito dall'argomento di una funzione/macro. L'esempio mostra anche lo scopo della segnatura delle funzioni: questa si aspetta quattro argomenti tra graffe, gli ultimi due indicano codice da eseguire se un test viene valutato vero o falso (rispettivamente T e F), in questo caso se c'è una corrispondenza tra il primo argomento e uno degli elementi nel secondo. Sono forniti anche `\str_case:nn`, `\str_case:nnT` e `\str_case:nnF` per i casi in cui il codice per i casi vero e falso non siano necessari: un buon esempio di ciò che considero un'interfaccia coerente.

Un altro esempio di interfaccia coerente. Supponiamo di dover fare qualcosa con due liste di token, una delle quali è esplicita e l'altra è talvolta solo disponibile come valore di una variabile *token list*.

```
\cs_new_protected:Nn \manual_foo:nn
{
  Do~something~with~#1~and~#2
}
\cs_generate_variant:Nn
  \manual_foo:nn
  { nV , Vn , VV }

\manual_foo:nn { Bar } { Foo }

\manual_foo:nV { Bar } \l_manual_whatever_tl

\manual_foo:Vn \l_manual_whatever_tl { Foo }

\manual_foo:VV
```

```
\l_manual_whatever_a_tl
\l_manual_whatever_b_tl
```

La *variante* è definita in modo coerente, ciò che ci dà quattro funzioni con nomi simili che eseguono la stessa azione su argomenti specificati diversamente. Fare lo stesso in L^AT_EX classico richiede giochi di prestigio con gli argomenti e qualche `\expandafter` ben piazzato: un gioco divertente, che però conduce spesso a indesiderate duplicazioni di codice. L'argomento di tipo V significa che l'argomento deve essere una variabile, il cui valore è estratto e sostituito con un argomento tra graffe per poi essere trattato dalla funzione di base.

Ecco i contro. Il codice è molto più verboso, come il codice C è molto più verboso dell'Assembler; è il prezzo da pagare quando ci sono molte più funzioni di base disponibili. Non si deve mai usare `\expandafter`, il giocattolo preferito di ogni programmatore T_EX; in particolare, non c'è più `\expandafter\@firstofone`.⁶ Scrivere codice `expl3` richiede ancora di capire come funziona l'espansione di macro e i messaggi di errore sono spesso molto criptici perché spesso vengono dal livello più basso di T_EX; va rilevato però che anche programmare in plain T_EX può portare a messaggi di errore incomprensibili.

7 I veri vantaggi

L'albero `.../tex/latex` in T_EX Live comprende più di 2000 cartelle. Se guardiamo dentro quei pacchetti, vediamo facilmente che reinventano la ruota molte volte. Non è affatto raro vedere reimplementate anche funzioni già disponibili nel nucleo di L^AT_EX. La macro `\ls` descritta prima, con la sua ausiliaria `\lsA` è un esempio: il codice è buono, ma può essere ritrovato in un numero indefinito di piccole variazioni, ogni volta che deve essere eseguita un'operazione di *parsing*. Lo stesso può dirsi di dozzine di costrutti tipici: avere un'ampia base di funzioni per le azioni di uso comune renderà il codice più leggibile e comprensibile. Studiare l'esempio di `\xyzy` è certamente divertente e istruttivo per afferrare concetti connessi con le macro ricorsive, in particolare la *tail recursion*. Ma perché rifare il lavoro che il L^AT_EX team ha già fatto per noi?

Gli articoli di Frank Mittelbach (con Rainer Schöpf o Chris Rowley) (MITTELBACH e SCHÖPF, 1991; MITTELBACH e ROWLEY, 1992, 1997) su L^AT_EX3 sono letture molto interessanti per capire gli scopi del progetto. Non è solo 'evitare gli spazi spuri' o 'non dimenticare spazi necessari': abbiamo a disposizione, in uno stato piuttosto maturo, un ambiente di programmazione quasi completo ed estendibile che ci libera dal dover pensare alle

6. Sto scherzando, è chiaro.

faccende di basso livello e permette di concentrarci sulla vera programmazione.⁷

Poiché il convegno del TUG si è svolto a Darmstadt, Germania, colgo l'occasione per chiudere in tedesco, con scuse a David Hilbert:

Aus dem Paradies,
das das L^AT_EX3 Team uns geschaffen,
soll uns niemand vertreiben können.

(Nessuno ci espellerà dal paradiso che il L^AT_EX3 team ha creato per noi.)

Riferimenti bibliografici

EIJKHOUT, V. (1992). *T_EX by Topic, A T_EXnician's Reference*. Addison-Wesley, Reading, MA, USA. <http://eijkhout.net/texbytopic/>.

KNUTH, D. E. (1986). *The T_EXbook*, volume A di *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA.

MITTELBACH, F. e ROWLEY, C. (1992). «L^AT_EX 2.09 $\hookrightarrow$ L^AT_EX3». *TUGboat*, **13** (1), pp.

7. In una conversazione durante il convegno del TUG, Stefan Kottwitz mi chiese di dire quali siano i più importanti passi avanti fatti da `expl3`; risposi 'i calcoli con virgola mobile e le espressioni regolari'. Sono ovviamente implementati con le primitive di T_EX, ma la mia opinione è che non sarebbero così potenti se `expl3` non fosse già stato sviluppato.

96–101. <http://tug.org/TUGboat/tb13-1/tb34mittl3.pdf>.

— (1997). «The L^AT_EX3 Project». *TUGboat*, **18** (3), pp. 195–198. <http://tug.org/TUGboat/tb18-3/13project.pdf>.

MITTELBACH, F. e SCHÖPF, R. (1991). «Towards L^AT_EX 3.0». *TUGboat*, **12** (1), pp. 74–79. <http://tug.org/TUGboat/tb12-1/tb31mitt.pdf>.

THE L^AT_EX PROJECT (2015a). «The L^AT_EX3 interfaces». <http://mirror.ctan.org/macros/latex/contrib/l3kernel/interface3.pdf>.

— (2015b). «The `expl3` package and L^AT_EX3 programming». <http://ctan.org/pkg/l3kernel>.

— (2015c). «The `l3regex` package: Regular expressions in T_EX». <http://ctan.org/pkg/l3regex>.

— (2015d). «The `xparse` package: Document command parser». <http://ctan.org/pkg/xparse>.

▷ Enrico Gregorio
Università di Verona
Dipartimento di Informatica
Strada le Grazie 15
Verona (Italy)

Easy T_EX automation with arara

Paulo Roberto Massa Cereda

Abstract

This article introduces **arara**, a T_EX automation tool based on rules and directives. The tool is an effort to provide a concise way to automate the daily T_EX workflow for users and also package writers. The author presents a quick tutorial on how **arara** works and unveils some of the new features and improvements of the upcoming version.

Sommario

Questo articolo introduce **arara**, uno strumento di automazione di T_EX basato su regole e direttive. Lo strumento risulta da uno sforzo di fornire un modo conciso per automatizzare la procedura di lavoro quotidiana degli utenti e dei programmatori dei pacchetti. L'autore presenta un rapido corso sul funzionamento di **arara** e svela alcune delle nuove caratteristiche e miglioramenti della versione in arrivo.

1 Prologue

Back in 2007, I had a complex T_EX project to tackle. Needless to say, I was very worried, not only because little I knew about the ways of the T_EX force, but the compilation steps required to achieve the final results were getting out of hand. Something had to be done, or this whole adventure was about to end in the worst possible way.

I tried **latexmk** first and it did a pretty good job. Sadly, my project had indeed a very complex compilation workflow and the tool was being pushed to its limits – my `.latexmkrc` file was getting more and more complicated as time went by. I then thought I should try a different solution.

1.1 Beware of lightning bolts

Suddenly, it was like being struck by a lightning bolt: I *knew* all the steps I had to reproduce beforehand, I only had to find a way to automate them! Trying to infer things from (at least) context-sensitive languages like T_EX would mean a lot of work! And all that guessing was unnecessary energy waste (although it could be a great excuse for eating more chocolate). So let us take the low resistance route.

So far, so good. I knew what I had to do, but how tell that to my potential new tool? Looking at how a compiler works, i.e. read a source file, ignore all comments and processes the rest, I thought to exploit the complement of it? My new tool would read instructions inside T_EX comments,



FIGURE 1: Do you like araras? We do, specially our tool which shares the same name of this colorful bird.

which would be ignored by conventional engines, thus causing no harm and side effects at all!

Time to make things happen. I sat in front of my computer and started to code while listening to Pink Floyd. In a couple of hours, I had a new tool tackling my T_EX project. But the adventure was not over.

1.2 I blame Enrico and Marco

I mentioned my typographic adventure in the chat room of the T_EX community at StackExchange and my good friend Enrico Gregorio encouraged me to release a public version of my newly created tool. So **arara** was introduced to the T_EX world. Figure 1 shows a lovely photo of the bird which inspired my tool's name.

The reception was surprisingly positive. Marco Daniel, a German T_EXnician and a good friend, liked the idea so much and became an evangelist. He helped me develop **arara**, writing code and providing feedback.

But I was not ready for a major release. I am far from being a perfectionist, but the code at the time was not acceptable yet. **arara** had just hatched (version 1.0) and required some code improvements before taking any further steps.

1.3 I blame Brent

It took me a lot of versions. When the counter stopped at version 3.0, Brent Longborough, **memoir** fanboy and a good friend, helped me with the code and the user manual. Then we decided it was time for **arara** to graduate and be released in T_EX Live. My life had changed.

It was a success. A lot of people liked the idea of explicitly telling **arara** how to compile their documents instead of relying on guesswork. But then, the inevitable happened: a lot of bugs had emerged from the dark depths of my code.

1.4 I blame Nicola

I was about to give up. My code was not awful, but there were a couple of critical and blocking bugs. Something very drastic had to be done in order to put **arara** back on track. I decided to rewrite the tool entirely from scratch. I created a sandbox and started working on the new code.

My good friend Nicola Talbot helped me with the new version, writing code, fixing bugs and suggesting new features. Soon, we all reached a very pleasant result. It was like **arara** was about to hatch again. Version 4.0 was at our hands.

1.5 The future

Sadly, version 4.0 is not yet in T_EX Live, and the reason for that is quite embarrassing: I had no time to work on the new user manual yet! Since the new 4.0 version ships with several features, improvements and bug fixes, it will take some time to cover them all. Hopefully, I will manage to finish this manual by the end of the year, so the new shiny version will soon make our users happy.

This article covers some new features and improvements of version 4.0 of **arara**. Of course, it would take more than a couple of pages to properly cover the tool usage, but at least this text will provide some insights of things to come. Thank you very much, dear reader, for joining my adventure. Have a glass of wine and enjoy the reading.

2 Two-minute tutorial

This section is a quick walkthrough on how **arara** works. As I have mentioned in the prologue, the tool takes a different approach compared to other tools on how to inspect T_EX code, and the key difference is that there is no guesswork. Let us say we have the following file **hello.tex** as an example:

```
\documentclass{article}
\begin{document}
Hello world!
\end{document}
```

Now, when running either **latexmk hello** or **rubber -pdf hello** in the terminal, we will get the expected result, but **arara hello** will not produce anything! In fact, **arara** will even complain about it:

```
It looks like no directives were
found in the provided file. Make
sure to include at least one
directive and try again.
```

What a rude bird! Wait a second, in fact, the tool is actually telling us what happened: there were no directives.

2.1 Directives

A directive is a special comment inserted in the source file in which you indicate how **arara** should behave. You can insert as many directives as you want and in any position of the file. Let us take a look at how a directive looks like:

```
% arara: <directive>
```

That looks interesting: so I can replace *<directive>* by any command I want, right? Suppose I want to run a command named **foo** on my previous file, so let us try:

```
% arara: foo
\documentclass{article}
\begin{document}
Hello world!
\end{document}
```

Surprisingly, **arara** will complain once again! The tool will always try to help us and describe exactly what has happened:

```
I could not find a rule named
'foo' in the provided rule paths.
Perhaps a misspelled word? I
was looking for a file named
'foo.yaml' in the following
paths in order of priority:
(/home/paulo/arara/rules)
```

According to the message, **arara** was not looking for a command, it was looking for a rule! This bird is surely naughty.

2.2 Rules

It is important to note that a directive is not the command to be executed, but the name of the rule associated with that directive! So there has to be a rule named **foo** in order for **arara** to understand what to do (once a directive is found, the tool will look for the associated rule).

Fortunately, **arara** is released with several rules out of the box, so it covers most common use cases. Should you wish to define your own rule, the user manual has an entire chapter dedicated to how to write them. In our previous example, we would have to come up with our own **foo** rule, but that is a story for another day (and another article).

Back to our document, how do we compile **hello.tex** with PDF_LA_TE_X? It is just a matter of including the right name of the rule to be invoked (in our case, **pdflatex**):

```
% arara: pdflatex
\documentclass{article}
\begin{document}
Hello world!
\end{document}
```

Now, after running **arara hello**, we will finally get the expected result, the very same we got from **latexmk hello** or **rubber -pdf hello**. Hooray!

2.3 Types of directives

Now that we have learned about rules and directives, it is time to dive deeper into the directive concept. In short, there are two types of directives in **arara**. The first one has already been mentioned, it has only the rule name (identifier):

```
% arara: <directive>
```

Sometimes, however, we need to provide additional information to the rule. That is reason for the second type, the parametrized directive, to exist.

As the name indicates, we have directive arguments! They are mapped by their identifiers and not by their positions. The syntax for a parametrized directive is:

```
% arara: <directive>: { <arguments> }
```

Each argument is defined according to the rule mapped by the directive. This means you cannot use an argument **foo** in a directive **bar** if the rule **bar** does not offer support for it (that is, **bar** has to have **foo** defined as argument in its list of arguments inside the rule scope). The syntax for an argument is:

```
<key> : <value>
```

Suppose we would like to enable shell escape for PDF_LA_TE_X when compiling **hello.tex**. We can achieve that by providing a parametrized directive, like this one:

```
% arara: pdflatex: { shell: yes }
\documentclass{article}
\begin{document}
Hello world!
\end{document}
```

Of course, the **shell** argument is defined in the **pdflatex** rule scope, otherwise **arara** would raise an error about an invalid argument key. If we try to inject a **foo** argument in the previous parametrized directive, we will get this message:

```
I have spotted an error in
rule 'pdflatex' located at
'/home/paulo/arara/rules'. I
found these unknown keys in the
directive: '(foo)'. This should
be an easy fix, just remove them
from your map.
```

As the message suggests, we need to remove the unknown argument key from our directive or rewrite the rule in order to include it. The first option is, of course, easier.

2.4 Overriding the default document

There is a reserved argument key named **file**, whose value is a list (represented by square brackets), which is available for every rule. If you want to override the default value of the main document for a specific directive, use this argument key on that directive and assign a list to it. For example, you if you want to run PDF_LA_TE_X on files **foo.tex** and **bar.tex** instead of the default **hello.tex**, we can use the following trick (there is a new feature in this code, but I will talk about it later on, in Subsection 3.5):

```
% arara: pdflatex: {
% arara: --> files: [ foo.tex, bar.tex ]
% arara: --> }
\documentclass{article}
\begin{document}
Hello world!
\end{document}
```

Note that **arara** will handle this occurrence as two invocations of PDF_LA_TE_X, one for each provided file (the terminal output and log file will tell you so).

2.5 Ready, set, go!

I believe the workflow is pretty straightforward. Of course, there is much more to **arara** than what I have described above, which covered enough to understand the basics and safely move on to the new features of version 4.0 and have fun. The user interface is one of the major selling points of **arara**, as it provides a clean and easy pattern for all users, from newbies to T_EXnicians.

3 New features and improvements

This section unveils some of the new features and improvements in version 4.0. As mentioned in Section 2, the tool usage remains the same, so do not worry with incompatibilities (apart from just one change for rule makers in the rule scope, but the fix is incredibly easy and with no impact whatsoever to the casual users).

3.1 Friendly and helpful messages

As noted in Section 2, messages were developed to be friendly and helpful. In version 4.0, we decided to make them even friendlier and include suggestions for correcting errors or improving usage. For, example, consider the following message:

```
I could not find a rule named
'foo' in the provided rule paths.
Perhaps a misspelled word? I
was looking for a file named
'foo.yaml' in the following paths
in order of priority:
(/home/paulo/arara/rules)
```

The message does not only tells you that the `foo` rule was not found, but it also provides information about all the search locations. This bird is actually quite clever! The next message is about an unknown argument key:

```
I have spotted an error in
rule 'pdflatex' located at
'/home/paulo/arara/rules'. I
found these unknown keys in the
directive: '(foo)'. This should
be an easy fix, just remove them
from your map.
```

This message tells us the rule, its location and a set of unknown argument keys. It also provides a suggestion as how to fix this issue, which involves removing the problematic argument from the directive map. Let us see yet another message:

```
I read a directive (1) and found
out that 'files' requires a list.
Please make sure to correct the
type to a proper list and try
again.
```

The message tells us that `files` was used as a parameter in the directive found in line 1 of the source code, but the type is not a list. The message also suggests how to fix this issue.

3.2 Improved command line layout

The command line layout was completely redesigned in order to look more pleasant to the eye (after all, we work with T_EX and friends). First and foremost, **arara** displays details about the file being processed, including size and modification status:

```
Processing 'hello.tex' (size: 132
bytes, last modified: 08/21/2015
11:09:04), please wait.
```

The list of tasks was also redesigned to be fully justified, and each entry displays both task and subtask names, besides of course the usual execution result. Figure 2 shows the new output. Note that **arara** also displays the execution time before terminating.

Verbose mode was also redesigned in order to avoid unnecessary clutter (there would be a lot of information already on screen), so it would be easier to spot each task. Error messages are now displayed in a box of their own, separated from the list of tasks.

3.3 Improved logging

The logging framework was also improved in order to better organize gathered information. The `arara.log` file contains the following enhancements (logging can be enabled through the `--log` flag):

1. System information is automatically collected in order to help debugging, such as the underlying operating system, architecture, Java virtual machine version and path, as well as the **arara** executable path.
2. Details about the file being processed, including size and modification status, are added to the logging handler (the very same information displayed in the terminal output).
3. The tool now includes all potential patterns found during the directive extraction phase, including the line number and content.
4. All directives, after being properly validated, are listed in the registry, including their potential arguments and conditionals.
5. Every task and subtask have an entry in the registry, containing the rule identifier and path, the corresponding system commands and output buffers, as well as the result status.

The resulting file is less cluttered and more useful when it comes to monitor and debug the automation process (it also helps me to guide users towards solving potential usage problems with **arara**).

3.4 Improved workflow

In previous versions, the workflow consisted of three steps: extract the directives from the source code, evaluate them beforehand and then execute all directives in batch mode. Starting with **arara** 4.0, we change this workflow to be based more on the REPL concept (read, eval, print, loop). In other words, directives are still extracted beforehand, but the tool will evaluate and execute one directive at a time before moving to the next one in the list.

This subtle change allows rules to be more context-aware, in the sense of inspecting the environment after the previous task has finished and take the appropriate actions, if needed.

3.5 Multiline directives

Sometimes, directives can span several columns of a line, particularly the ones with several arguments or conditionals. From **arara** 4.0, we can split a directive into multiple lines by adding `% arara: -->` to each line which should compose the directive (there is no need of them to be in contiguous lines, though):

```
% arara: pdflatex: {
% arara: --> shell: yes,
% arara: --> synctex: yes
% arara: --> }
```

It is worth mentioning that multiline support works with conditionals as well (described later on, in Subsection 3.9):

```

/_ ' | , _/_ ' | , _/_ ' |
| ( | | | | ( | | | | ( | |
\_ , _ | | \_ , _ | | \_ , _ |

```

Processing 'hello.tex' (size: 132 bytes, last modified:
08/21/2015 11:09:04), please wait.

(PDFLaTeX) PDFLaTeX engine SUCCESS
(Clean) Cleaning feature SUCCESS
(Clean) Cleaning feature SUCCESS

Total: 0.32 seconds

FIGURE 2: Improved command line layout in **arara**.

```
% arara: pdflatex
% arara: --> if missing('pdf')
% arara: --> || changed('tex')
```

3.6 --dry-run mode

Another interesting feature is the inclusion of a `--dry-run` flag, which goes through all the work of reading the tasks and subtasks, but without actually executing them. For example, `arara hello -n` (the long option `--dry-run` also works) will display the following result:

[DR] (PDFLaTeX) PDFLaTeX engine

Authors: Marco Daniel, Paulo Cereda
About to run: [pdflatex, hello.tex]

Note that the rule authors are displayed (so they can be blamed in case anything goes wrong), as well as the system command to be executed. It is an interesting approach to see everything that will happen to your document and in which order. It is important to observe, though, that conditionals are not evaluated in this mode.

3.7 Support for lists and maps as values

In previous versions, only `files` could be a list (Subsection 2.4). From version 4.0 on, directive arguments can hold any type of value, from single elements to maps and lists (of course, the rule has to be prepared to handle a list or a map, or even a combination of both). For example, the new `clean` rule allows us to specify a list of extensions to be removed (Figure 2 also shows the output of this execution):

```
% arara: pdflatex
% arara: clean: {
% arara: --> extensions: [ aux, log ]
% arara: --> }
\documentclass{article}
\begin{document}
```

```
Hello world!  
\end{document}
```

There are several helper methods available in the rule scope to ease the use of lists and maps, so rule makers can have fun with these data types.

3.8 File hashing

arara 4.0 features four methods for file hashing in the rule and directive scopes, presented as follows:

- **changed(extension)**: checks if the file's base-name concatenated with the provided extension has changed its checksum from last verification.
- **changed(file)**: the very same idea as the previous method, but with a proper `File` object instead.
- **unchanged(extension)**: checks if the file's base-name concatenated with the provided extension is unchanged from last verification. It is the opposite of the **changed(...)** method.
- **unchanged(file)**: the very same idea as the previous method, but with a proper `File` object instead.

The value is stored in a file named `arara.xml` as a pair containing the file path and its corresponding CRC-32 hash (the file is created if missing). If the entry already exists, the value is updated, or created otherwise.

3.9 Conditionals

One of the most awaited features that version 4.0 introduces is the support of conditionals, that is, logic expressions processed at runtime in order to determine whether and how the directive should be processed. The following types are allowed:

- **if:** evaluated beforehand, the directive is interpreted if and only if the result is true.

- **unless**: evaluated beforehand, the directive is interpreted if and only if the result is false.
- **until**: directive is interpreted the first time, then the evaluation is done; while the result is false, the directive is interpreted again and again.
- **while**: evaluated beforehand, the directive is interpreted if and only if the result is true, and the process is repeated while the result still holds true.

It is important to observe that **arara** has a maximum number of loops, that is, we do not want a directive to be executed forever. The default value is set to 10, but it can be overridden either in the configuration file (through the **loops** key) or in the command line flag (through the **--max-loops** flag).

Several methods are available in the directive scope in order to ease the writing of conditionals, including (file hashing methods are available as well):

- **exists(extension)**: checks whether the file's basename concatenated with the provided extension exists. For example, if the file is named **foo.tex**, **exists('pdf')** will check whether **foo.pdf** exists.
- **exists(file)**: checks whether the file exists. Note that this is a proper call to the **File** object.
- **missing(extension)**: the opposite of **exists**, checks whether the file's basename concatenated with the provided extension does not exist.
- **missing(file)**: checks whether the file does not exist. As mentioned before, this is a proper call to the **File** object.
- **found(extension, regex)**: checks whether the file's basename concatenated with the provided extension matches the provided regular expression.
- **found(file, regex)**: the very same idea as the previous method, but with a proper **File** object instead.

Logical connectors can be used in order to create boolean expressions: **&&** (and), **||** (or), and **!** (negation). The following example runs PDF_LA_TE_X if there is no resulting PDF file or if the source code has changed from last execution:

```
% arara: pdflatex
% arara: --> if missing('pdf')
% arara: --> || changed('tex')
```

The next example tells **arara** to run PDF_LA_TE_X until the corresponding log file does not contain undefined references anymore:

```
% arara: pdflatex
% arara: --> until !found('log',
% arara: --> 'undefined references')
```

As mentioned, a lot of methods are available in the directive scope in order to provide a better experience and consistent execution, but I cannot list them all here. Please refer to the user manual and check the complete reference (when, of course, I finally manage to finish it).

4 Closing remarks

It took me so long to finally fix bugs and implement new features and enhancements, but I strongly believe the final result made the wait truly worthwhile. **arara** has become mature with time, and so has the code base. It is wonderful to look back at when I started and where we are today. I owe everything to my team and also to the hundreds of friends out there who decided to give this humble tool a try. Thank you very much.

arara is already available in T_EX Live and also as a standalone tool, but note that the version covered in this article is not officially released yet. The tutorial presented in Section 2 still applies to version 3.0, but the new features and improvements presented in Section 3 really require at least version 4.0, which is yet only available in my source code repository:

<https://github.com/cereda/arara>

Hopefully, I will soon be able to finish the user manual and finally release version 4.0 to the entire T_EX world. Feel free to contribute to the project by submitting bugs, sending pull requests or maybe by translating the tool to your language. And if you want to support the L^AT_EX development with a donation, the best way to do this is by donating to your local T_EX Users Group.

Happy T_EXing with **arara** 4.0!

Acknowledgments

The author wishes to thank Enrico Gregorio, Francesco Endrici and all friends from $\mathcal{U}\mathcal{I}\mathcal{R}$ for the opportunity of writing this humble article for *Ar_sT_EXnica*.

▷ Paulo Roberto Massa Cereda
San Paolo University, Brasil
paulo.cereda@usp.br

Animazioni e grafici interattivi in $\text{\LaTeX}^*$

Grazia Messineo, Salvatore Vassallo

Sommario

In questo articolo illustriamo alcuni pacchetti $\text{\LaTeX}$ per la creazione di animazioni e grafici interattivi. Inoltre, mostriamo il loro utilizzo a livello didattico, in presentazioni, con il pacchetto `beamer`, e all'interno degli esercizi di autovalutazione on line prodotti con il pacchetto `minerva`.

Abstract

We present some $\text{\LaTeX}$ packages to create animations and interactive plots. We present also their use for educational applications, both in presentations, realized with the `beamer` package, and in the on line exercises created with the `minerva` package.

1 Introduzione

Quando si realizzano applicazioni didattiche (slides, esercizi di autovalutazione, ecc.) capita spesso che l'introduzione di animazioni o grafici interattivi faciliti la comprensione di un concetto. Si pensi, ad esempio, in ambito matematico, a quando si vuole illustrare come varia il grafico di una parabola al variare dei parametri.

In questo lavoro, passeremo rapidamente in rassegna alcuni strumenti esterni che consentono di realizzare animazioni in $\text{\LaTeX}$, per poi introdurre alcuni pacchetti $\text{\LaTeX}$ che permettono di creare pdf animati o interattivi.

Infine, mostreremo alcune applicazioni, all'interno di presentazioni realizzate con il pacchetto `beamer` e all'interno di esercizi interattivi di autovalutazione, ottenuti con il pacchetto `minerva` (MESSINEO e VASSALLO, 2013).

Nei prossimi paragrafi faremo alcuni esempi di ciò che è possibile realizzare con gli strumenti presentati. L'interattività dei file spesso rende del tutto inutile l'inserimento di immagini in un articolo. Per la visualizzazione degli esempi qui descritti rimandiamo alla presentazione realizzata per il convegno.

2 I programmi esterni

In questo paragrafo presentiamo brevemente alcuni programmi esterni, liberamente disponibili in rete,

*Il lavoro si inserisce nel progetto di ricerca di interesse d'Ateneo dell'Università Cattolica "Progetto M.In.E.R.Va.: Creazione di esercizi di tipo interattivo con percorsi differenziati per il recupero delle carenze e la valorizzazione delle eccellenze in matematica".

con i quali è possibile realizzare grafici interattivi da importare in $\text{\LaTeX}$.

2.1 R

R è un software per la manipolazione di dati, il calcolo e la visualizzazione grafica. È orientato alla statistica.

La sua struttura è simile a quella di $\text{\LaTeX}$: al centro del sistema vi è il nucleo, nel quale sono descritte le regole del linguaggio, che è modificabile solo dagli sviluppatori. Il nucleo è circondato da un primo anello nel quale è contenuta una serie di funzionalità di base (gli algoritmi che consentono di costruire le funzioni statistiche). Più all'esterno sono contenuti tutti i pacchetti che coprono molte esigenze di tipo statistico.

R può essere usato insieme a $\text{\LaTeX}$ mediante l'uso di alcuni pacchetti specifici, dei quali il più conosciuto è `Sweave`, presentato nel convegno $\text{\LaTeX}$ 2014 (BALTIERI e SCANDOLA, 2014).

Le animazioni in $\text{\LaTeX}$ possono essere create in due modi:

- inserendo direttamente il codice R nel file $\text{\LaTeX}$ e usando per la compilazione il pacchetto `Sweave` e per le animazioni il pacchetto `animate` (v. par. 3.1.1);
- creando in R dei file di immagine che poi vengono importati nel file $\text{\LaTeX}$ e animati sempre con l'uso del pacchetto `animate`.

2.2 Asymptote

`Asymptote` è un linguaggio grafico vettoriale molto potente che mette a disposizione dell'utente un ambiente basato su coordinate per il disegno tecnico. Le etichette e le equazioni sono scritte in $\text{\LaTeX}$, in modo da generare un output PostScript di alta qualità.

`Asymptote` può essere usato direttamente in $\text{\LaTeX}$ attraverso il pacchetto omonimo, che mette a disposizione due ambienti fondamentali:

- `asydef`, che contiene le definizioni globali del documento;
- `asy`, che contiene il codice `Asymptote` vero e proprio.

Il file pdf è ottenuto con i seguenti comandi:

```
pdflatex filename
asy filename-1.asy
pdflatex filename
```

I grafici che si ottengono possono essere sia statici che interattivi (l'utente può ruotarli, ecc.).

La realizzazione di grafici ottenibili programmando **Asymptote** è stata illustrata durante il convegno **GJt 2009**. (DE MARCO, 2009)

2.3 Sage

Sage è un programma multiplatforma, scritto in Python, orientato alla matematica, che consente di risolvere problemi di varia natura (analisi, calcolo combinatorio, ecc.).

Il programma è disponibile per Linux e per Mac Os X; per Windows è stata implementata una versione che può essere utilizzata insieme alla Oracle VirtualBox. Esiste anche la possibilità di utilizzarlo (anche insieme a **LATEX**) in cloud (<https://cloud.sagemath.com/>).

Sage può essere utilizzato con **LATEX** in vari modi.

Ad esempio, il pacchetto **sagetex** (DRAKE *et al.*, 2009) consente di usare codice **sage** nel documento e fa sì che questo codice sia correttamente compilato durante la compilazione **LATEX**. Il codice può essere interpretato sia con un'installazione locale di **sage**, sia utilizzando un'installazione su server remoto, attraverso uno script Python.

Esiste anche il programma **sws2tex** (HUSS e MAŘÍK, 2009), un convertitore in grado di trasformare i file **sws** di **sage** in file **tex** e di compilarli, producendo in output un file **pdf**.

Sage consente di realizzare animazioni e grafici interattivi di vario tipo, che possono essere salvati in **pdf** e utilizzati all'interno di file **LATEX**.

3 I pacchetti **LATEX** per le animazioni

In questo paragrafo presentiamo le strategie che si possono usare in **LATEX** per la realizzazione di grafici animati e alcuni pacchetti che consentono di ottenerli. Tali pacchetti permettono di ottenere grafici interattivi con o senza l'ausilio di programmi esterni.

Tutti i grafici animati realizzati con questi pacchetti vengono visualizzati correttamente con Adobe Acrobat Reader e, in alcuni casi, con altri visualizzatori (ad esempio, PDF-XChange in ambiente Windows).

3.1 Le strategie per ottenere grafici animati

In **LATEX** è possibile ottenere grafici animati usando due strategie:

- realizzare, mediante codice **LATEX** o programmi esterni, delle immagini in formati compatibili, da inserire nel file principale attraverso l'uso di appositi pacchetti: **animate** (v. par. 3.1.1) per tutti i file di immagine compatibili con il motore usato per la compilazione; **movie15** se si vogliono inserire filmati o, in

particolare, file di immagine in formato **gif**, come le **gif** animate (v. par. 3.1.2).

- inserire direttamente il codice che genera l'immagine all'interno di un ambiente in grado di generare contemporaneamente le immagini e la loro animazione: tale obiettivo viene raggiunto con il pacchetto **animate**.

Nei prossimi paragrafi presenteremo i due pacchetti utili per la realizzazione di animazioni.

3.1.1 *Animate*

Il pacchetto **animate** di Alexander Grahn (GRAHN, 2015a) consente di creare file **pdf** con contenuti animati, ottenuti da un insieme di file grafici o file immagine, così come da grafici generati con **PSTricks** o **TikZ**, oppure semplicemente dal testo. Permette di creare animazioni anche con grafici vettoriali.

I formati immagine accettati sono tutti quelli compatibili con il motore scelto per la compilazione: **eps**, **ps** e **mps** se si usa **LATEX**, **dvips** e **ps2pdf**; **pdf**, **mps**, **png**, **jpg**, ecc. se si usa **PDFLATEX**.

Il pacchetto mette a disposizione due strumenti fondamentali.

- Il comando **\animategraphics**, che consente di creare animazioni da insiemi di file grafici creati esternamente oppure da file **pdf** che contengono più pagine. La sintassi del comando è

```
\animategraphics [<opzioni>]
{<frame rate>}
{<nome di base del file>}
{<primo>}{<ultimo>}
```

Tra le opzioni: può essere indicata la *timeline*, scritta su un file esterno che viene successivamente caricato; può essere assegnata un'etichetta (univoca); può essere impostata una pausa automatica, ecc.

Il parametro **frame rate** indica il numero di fotogrammi che vengono visualizzati in un secondo; **nome di base del file** indica la parte comune dei nomi assegnati ai file di immagine da usare per l'animazione (ad esempio, se i file di immagine sono **file01.eps**, **file02.eps**, ecc., **nome di base del file** è **file**); **primo** e **ultimo** sono, rispettivamente, il primo e l'ultimo numero assegnato ai file di immagine.

- L'ambiente **animateinline**, che consente di creare animazioni dal codice racchiuso al suo interno. Il codice all'interno può essere creato mediante l'ambiente standard di **LATEX**, **picture**, oppure usando i pacchetti **pstricks** (par. 3.2.2) o **tikz/pgf** (par. 3.2.3).

La sintassi dell'ambiente è

```

\begin{animateinline}
[<opzioni>]{<frame rate>}
\dots codice ...
\newframe[<frame rate>]
\dots codice ...
\newframe*{<frame rate>}
\dots codice ...
\newframe
\multiframe{<numero di fotogrammi>}
{[<variabili>]}
{...codice ripetuto...}
\end{animateinline}

```

Le opzioni dell'ambiente sono simili a quelle del comando `\animategraphics`, così come il significato del parametro `frame rate`.

Il comando `\newframe` serve per chiudere il frame precedente ed iniziare uno nuovo; la versione con `*` prevede una pausa dopo il frame dietro al quale è posizionato. L'animazione riprende cliccando con il mouse.

Il comando `\multiframe` consente di costruire dei loop con le figure. Per creare il loop, si usa il parametro opzionale `variabili`, che contiene una lista di variabili (separate da virgola) che possono essere inizializzate ad un determinato valore e incrementate di un valore predefinito e fisso.

Si rinvia alla documentazione del pacchetto (GRAHN, 2015a) per ulteriori dettagli sulle opzioni del comando e dell'ambiente e anche per alcuni esempi di utilizzo.

3.1.2 Movie15

Il pacchetto `movie15` di Alexander Grahn (GRAHN, 2012) consente di inserire all'interno di un file $\text{\LaTeX}$ immagini in formato `gif`, in particolare `gif` animate.

Il pacchetto è, per la verità, obsoleto e lo stesso autore consiglia l'uso del pacchetto più recente `media9` (GRAHN, 2015b), che però non consente di inserire direttamente `gif` animate.

Il pacchetto mette a disposizione, per il caricamento del file, il seguente comando:

```

\includemovie[<opzioni>]{<larghezza>}
{<altezza>}{<file>}

```

Nelle opzioni è possibile specificare se la riproduzione e l'arresto del file devono essere automatici, se si vuole che il file sia aperto in un visualizzatore esterno, ecc.

Se non si desidera o non si può usare `movie15`, è necessario convertire la `gif` animata in uno dei formati supportati da `animate` o da `media9` e usare uno dei due pacchetti.

3.2 I pacchetti per la realizzazione di immagini

In questo paragrafo presentiamo tre pacchetti: il pacchetto `multido`, che consente di automatizzare alcune operazioni ripetute e due pacchetti per la realizzazione di immagini, `pstricks` e `tikz/pgf`.

Non ci addentreremo in questa sede in spiegazioni dettagliate sul funzionamento dei pacchetti, ci limiteremo ad illustrare come possono essere adoperati per la creazione di immagini da rendere "animate".

3.2.1 Il pacchetto multido

Il pacchetto `multido` (VAN ZANDT, 2010) di T. Van Zandt, creato inizialmente per il pacchetto `pstricks`, ma utilizzabile anche indipendentemente da esso, mette a disposizione il comando omonimo, che consente di automatizzare una serie di operazioni ripetitive mediante un'unica istruzione.

La sua sintassi è:

```

\multido{variabili}{ripetizioni}{codice}

```

Il primo parametro obbligatorio, `variabili`, consente di dichiarare le variabili (separate da virgola) da usare nel disegno. Devono essere definite nella forma `variabile=valore iniziale+incremento` e l'incremento deve essere preceduto dal segno `-` se si desidera decrementare la variabile.

Il parametro `ripetizioni` indica il numero di volte in cui si vuole ripetere un'azione. Il parametro `codice` contiene invece il codice che si vuole ripetere.

Tale comando risulta particolarmente utile, tra le altre cose, per la produzione di file di immagine da rendere animati.

3.2.2 PSTricks

`PSTricks` è una collezione di macro di interfaccia tra $\text{\TeX}$ (e $\text{\LaTeX}$) e il linguaggio `PostScript`.

Consente di disegnare (quasi) tutti i tipi di oggetti, grafici 2D e 3D, a colori e non, e molto altro.

Usando `PSTricks`, un qualsiasi disegno viene "tracciato" attraverso una serie di comandi.

Poiché `PSTricks` è un insieme di macro in linguaggio `PostScript`, per visualizzare l'output correttamente si hanno due opzioni:

- compilarlo con $\text{\LaTeX}$, `dvips`, `ps2pdf`;
- usare il pacchetto `auto-pst-pdf` oppure passare l'opzione `pdf` al pacchetto (che chiama `auto-pst-pdf`). Ciò consente la compilazione diretta dei file contenenti codice `PSTricks` con `PDF $\text{\LaTeX}$` .

Il pacchetto `PSTricks` è molto ben documentato: in rete sono presenti diversi manuali e tutorial, ai quali si rimanda per una descrizione delle sue funzionalità di base e avanzate.

`PSTricks` consente di realizzare animazioni in due modi.

- Creando dal codice `PSTricks` file `pdf` con immagini interattive (dalle quali si possono poi ottenere `gif` animate, da usare in siti web o in altri programmi): in questo caso, si può usare il comando `\multido` già descritto nel par. 3.2.1.

Si consideri il seguente (semplicissimo) esempio:

```
% nel preambolo
\usepackage{multido,fp}
\usepackage{pstricks-add}
...
\multido{\n=0+1}{20}{
\begin{pspicture}(-3,-3)(3,3)
\ifodd\n
\FPeval\r{round((\n/10):1)}
\pscircle[linecolor=yellow]{\r}
\else
\FPeval\r{round((\n/10):1)}
\pscircle{\r}
\fi
\end{pspicture}
}
```

Questo codice produce una serie di immagini composte da cerchi di dimensione crescente colorati alternativamente di giallo e di nero. Per ottenere cerchi di dimensione decrescente, è sufficiente definire la variabile $\n$ a partire dal valore massimo e decrementarla anziché incrementarla.

Una volta creato il file pdf, è possibile inserirlo direttamente in un file $\LaTeX$ usando il comando `\animategraphics` del pacchetto `animate` (3.1.1). In alternativa, è possibile ottenere una gif animata (da inserire ad esempio in una pagina html o in presentazioni realizzate non in $\LaTeX$) usando `ImageMagick` e il comando

```
convert -delay x -loop 0
-density y -scale z
-alpha remove
file.pdf file.gif
```

La gif animata così ottenuta può essere utilizzata in un file $\LaTeX$ con l'ausilio del pacchetto `movie15`.

- È possibile anche inserire il codice per creare le immagini all'interno dell'ambiente `animateinline` (par. 3.1.1), come mostrato nel seguente esempio tratto da <https://www.tug.org/PSTricks/main.cgi?file=Animation/basics>:

```
\newwrite\TimelineFile
\immediate\openout
\TimelineFile=sinus.txt
\immediate\write
\TimelineFile{::0x0,1}%
\multido{\i=2+1}{90}{%
\immediate\write\TimelineFile{%
::\i}
}
\immediate\closeout\TimelineFile
```

Queste istruzioni servono per scrivere automaticamente il contenuto del file della timeline.

```
\psset{xunit=\pstRadUnit,
dashadjust=false}
\begin{animateinline}
```

```
[controls,timeline=sinus.txt,
begin={\begin{pspicture}
(-0.5,-1.5)(6.6,2)},
end={\end{pspicture}}]{10}
\psaxes[trigLabels,trigLabelBase←
=3]{->}(0,0)(-0.2,-1.5)(6.5,1.5)
[$t$, -90]{$y=\sin(t)$,0}
\psplot[linestyle=dashed,xunit=1cm,
algebraic]{0}{\psPiTwo}{\sin(x)}
\newframe
\multiframe{91}{n=0+4}{
\psset{xunit=1cm,linecolor=blue}
\pscustom[xunit=1cm,fillcolor=blue,
fillstyle=solid,linestyle=none,
algebraic,dimen=inner]{%
\psplot{0}{\n\space DegtoRad}
{\sin(x)}
\psline(!\n\space DegtoRad 0)}
\psplot[linestyle=dashed,xunit=1cm,
linecolor=black,algebraic]{0}
{\n\space DegtoRad}{\sin(x)}
\psdot[opacity=0.4,dotsize=3mm]
(!\n\space dup DegtoRad exch sin)
\psline[linestyle=dashed]
(!\n\space dup DegtoRad exch sin)
(!\n\space DegtoRad 0)}
\end{animateinline}
```

Con queste istruzioni si crea l'animazione inserendo il codice per l'immagine direttamente all'interno dell'ambiente e si inseriscono i pulsanti che permettono di controllarla. Il grafico che ne risulta è quello della funzione $y = \sin x$, con un punto che si muove su di esso e l'area compresa tra il grafico della funzione, l'asse x e la retta $x = x_0$, con x_0 ascissa del punto mobile, che si colora automaticamente.

3.2.3 TikZ/Pgf

Il pacchetto `pgf` e la sua interfaccia (non grafica) `TikZ` costituiscono un altro approccio alla grafica in $\LaTeX$. Infatti permettono di costruire figure mediante comandi simili a quelli di altri programmi o pacchetti, e di avere come output sia un file PostScript, sia un file pdf, senza l'intervento di programmi esterni, ma con l'usuale compilazione con $\LaTeX$ o $\PDF\LaTeX$. Il fatto di essere compatibile con $\PDF\LaTeX$ ha però l'inconveniente di una minore potenza del pacchetto rispetto a `PSTricks` in quanto il linguaggio PDF è meno potente (soprattutto da un punto di vista matematico) del linguaggio PostScript.

L'autore dei pacchetti è Till Tantau (autore anche del pacchetto `beamer`). La loro documentazione (TANTAU, 2013) è molto ricca.

Per usare il pacchetto `tikz` è necessario caricarlo direttamente, oppure caricare il pacchetto `pgf`. Per particolari scopi, si possono caricare altre librerie specializzate di `pgf`, ad esempio per disegnare diversi tipi di frecce o per usare sfondi, ecc.

Presenteremo qui solo alcuni esempi di utilizzo del pacchetto `pgf` per produrre grafici animati. L'idea di base è la stessa già illustrata per il pacchetto `pstricks` (v. 3.2.2).

- La prima possibilità è quella di creare file di immagine in formato pdf, o altro formato riconosciuto da L^AT_EX e successivamente utilizzare il comando `\animategraphics` del pacchetto `animate` per creare l'animazione.

Ad un risultato simile si può pervenire usando, insieme a TikZ, il pacchetto `tkz-fct`, che si basa sul programma di grafica Gnuplot (esempio tratto da <http://tex.stackexchange.com/questions/106140/definite-integral-animation-examples>):

```
% nel preambolo
\usepackage{tkz-fct}
\usepackage{multido}
...
\multido{\i=2+1}{20}{%
\begin{tikzpicture}[scale=1.25]
\tkzInit[xmax=8,ymax=4]
\tkzAxeXY[ticks=false]
\tkzGrid
\tkzFct[color=red,
domain=0.125:8]{4./x}
\tkzDrawRiemannSumInf[fill=green,
opacity=.2,color=green,interval=1:8,
line width=1pt,number=\i]
\end{tikzpicture}}
```

Le immagini ottenute con questo documento possono poi essere animate usando il comando `\animategraphics`.

Per ottenere le immagini da inserire nel comando `\animategraphics` è possibile utilizzare il pacchetto `pgfplots`, che illustreremo più dettagliatamente nel paragrafo 3.3.2. Tale uso è mostrato in questo esempio:

```
\foreach \pos in {0,0.05,...,1.05}{
\begin{tikzpicture}
\begin{axis}[grid= major,xlabel=$x$,
ylabel=$y$,
ylabel style={rotate=-90},
no marks]
\addplot{\x^2}
node[fill=orange,draw=blue,circle,
inner sep=1pt,pos=\pos]{};
\end{axis}
\end{tikzpicture}}
```

- La seconda possibilità è creare l'immagine all'interno dell'ambiente `animateinline`, come viene fatto in questo esempio tratto da <http://www.texample.net/tikz/examples/animated-definite-integral/>:

```
\newcounter{angle}
\setcounter{angle}{0}
\newcounter{r}
\newcommand{\escalar}[1]{
\setcounter{r}{#1 * #1 * #1}
\newcounter{m}
\setcounter{m}{0}
\newcounter{mc}
\begin{animateinline}[loop,controls,
poster= first,palindrome]{2}
\whiledo{\them < 21}{
\begin{tikzpicture}[scale=1.25]
```

```
\draw[red,thick,<->] (-1,1)
parabola bend (0,0) (2.1,4.41)
node[below right] {$y=x^2$};
\draw[loosely dotted] (-1,0)
grid (4,4);
\draw[->] (-0.2,0) -- (4.25,0)
node[right] {$x$};
\draw[->] (0,-0.25) -- (0,4.25)
node[above] {$y$};
\foreach \x/\xtext in {1/1, 2/2, ←
3/3}
\draw[shift={(\x,0)}] (0pt,2pt) --
(0pt,-2pt) node[below] {$\xtext$};
\foreach \y/\ytext in
{1/1, 2/2, 3/3, 4/4}
\draw[shift={(\x,\y)}] (2pt,0pt) --
(-2pt,0pt) node[left] {$\ytext$};
\setcounter{mc}{\value{m}*\value{m}}
\shade[top color=blue,
bottom color=gray!50](0,0)
parabola(0.1*\them,0.01*\themc)
|- (0,0);
\escalar{\them}
\draw (3cm,2pt) node[above]
{$\displaystyle\int_0^{\them/10}
!\!x^2\mathrm{d}x =
\displaystyle\frac{\ther}{3000}$};
\draw[fill=orange,color=orange]
(0.1*\them,0.01*\themc) circle(.5pt);
\end{tikzpicture}
\stepcounter{m}
\ifthenelse{\them < 21}{
\newframe
}
{\end{animateinline}}\relax
}
```

3.3 I grafici interattivi

In questo paragrafo presentiamo due pacchetti per la creazione di grafici interattivi: `interactiveplot` e `pgfplots`.

Definiamo grafici interattivi quei grafici dei quali l'utente può modificare in vari modi il comportamento (ad esempio variando, mediante pulsanti, il valore di uno o più parametri) o che, cliccando con il mouse, mostrano informazioni di vario tipo relative all'oggetto disegnato (ad esempio, le coordinate di un punto).

3.3.1 Interactiveplot

`Interactiveplot` è un pacchetto realizzato da R. Bock, P. Linares e J. Toro (BOCK *et al.*, 2014) che consente di creare, all'interno di un file pdf, grafici parametrici in due e tre dimensioni. Il comportamento del grafico al variare dei parametri può essere controllato da bottoni che li fanno aumentare o diminuire secondo un passo prefissato.

Un grafico 2D viene inserito nell'ambiente `iplotdd`, un grafico 3D nell'ambiente `iplotddd`.

All'interno dei due ambienti, l'espressione algebrica viene inserita come argomento del comando `\iplot` e l'interattività gestita attraverso alcune delle sue opzioni.

Per i grafici 2D, la sintassi è

```
\begin{iplotdd}[width=w,height=h]
\iplot[var={nome_v,vmin,vmax},
```

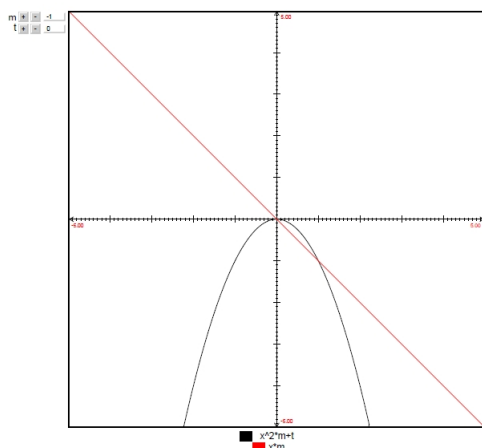


FIGURA 1: Una parabola e una retta variabili

```
param={nome_p,valore_iniz,step,min,max},
frange={nome,min,max},color=colore]
{funzione}
\end{iplotdd}
```

L'opzione `var` consente di definire la variabile indipendente e i suoi valori minimo e massimo. Il grafico sarà disegnato nell'intervallo $(vmin, vmax)$. L'interattività è prodotta dall'opzione `param`, mediante la quale vengono definiti uno o più parametri che compaiono nell'espressione algebrica della funzione. Ogni parametro è definito da un nome (`nome_p`). Deve essere attribuito un valore iniziale (il primo che compare sul grafico e per il quale esso è disegnato), il passo con cui il parametro deve essere diminuito o aumentato, il valore minimo e massimo. Se non vengono definiti parametri, il grafico diventa statico. Per ogni parametro, vengono inseriti nel file pdf due bottoni, uno con un segno `+` e l'altro con un segno `-`, che consentono di aumentare o diminuire il valore del parametro di `step` unità. L'opzione `frange` consente di definire la variabile dipendente e il suo intervallo di variazione.

La sintassi per i grafici 3D è identica, con l'unica differenza che devono essere definite due variabili indipendenti (sempre con l'opzione `var`).

Nello stesso ambiente possono essere inseriti più grafici dello stesso tipo (2D o 3D). È anche possibile usare lo stesso parametro per due o più grafici: in questo caso, la modifica del valore del parametro mediante i pulsanti `+` e `-` modifica l'aspetto di tutti i grafici che contengono quel parametro. Nell'esempio che segue, lo stesso parametro viene usato sia nel primo che nel secondo grafico (si veda la figura 1):

```
\begin{iplotdd}[width=400,height=400]
\iplot[var={x,-5,5},
param={m,-1,1,-10,10},
param={t,0,1,-10,10},
color=black,
frange={y,-5,5}]
{x^2*m+t}
\iplot[var={x,-5,5},
```

```
param={m,-1,1,-10,10},
color=red,
frange={y,-5,5}]
{x*m}
\end{iplotdd}
```

L'interattività dei grafici è garantita dall'inserimento di codice Javascript nel file pdf.

Il pacchetto ha buone potenzialità, ma al momento "soffre" di alcune limitazioni:

- non è possibile controllare la posizione dei pulsanti che animano il grafico;
- non è possibile controllare la posizione e lo stile della legenda dei grafici disegnati. Inoltre, i parametri sono sempre indicati con il loro nome e non vengono sostituiti i valori man mano loro attribuiti;
- non è possibile inserire punti, etichette, ecc.
- il pacchetto non funziona con le principali classi utilizzate per produrre presentazioni (beamer, powerdot, ecc.).

3.3.2 Pgfplots

Il pacchetto `pgfplots`, basato sul pacchetto `pgf` descritto nel paragrafo 3.2.3, è stato scritto da Christian Feuersänger, che ne cura anche la documentazione (FEUERSÄNGER, 2015).

Una descrizione di base molto dettagliata delle funzionalità del pacchetto è stata presentata al convegno `qJr` del 2011 (DE MARCO e GIACOMELLI, 2011).

Anche in questo caso presentiamo solo alcune applicazioni dei comandi del pacchetto alla produzione di grafici interattivi, rimandando al manuale e all'articolo per ogni altra esigenza.

Presentiamo due applicazioni realizzate con `pgfplots`.

- La prima è un grafico che mostra come varia il comportamento della parabola $y = ax^2$ al variare del parametro a :

```
% nel preambolo
\usepackage{pgfplots}
\usetikzlibrary{ocgx}
\tikzset{ocg button/.style={circle,
inner sep=.25em,switch ocg
with mark on={#1}{}}}
\tikzset{base/.style={
baseline=-0.5ex}}
\newcommand{\function}{\x^2}
\newcommand{\button}[2]{\tikz[base]
\node[fill=#2!30,ocg button=#1]{};
\newcommand{\plotit}[2]
{\addplot+[ocg={name=#2,ref=#2}]
{#1};\label{#2}}
\newcommand{\legendit}[3]
{\item[\ref{#1}]
#2x^2$ \button{#1}{#3}}
...
\begin{minipage}[b][0.4\textheight]
[1][0.7\textwidth]
\begin{tikzpicture}
```

```

\begin{axis}[grid= major,xlabel=$x$,
ylabel = $y$,
ylabel style={rotate=-90},
cycle list={blue,red,green!50!lime,
orange,cyan!50!blue},]
\plotit{\function}{first}
\plotit{1.5*\function}{second}
\plotit{2.25*\function}{third}
\plotit{3*\function}{fourth}
\plotit{5*\function}{fifth}
\end{axis}
\end{tikzpicture}
\end{minipage}
\begin{minipage}[b][0.5\textheight]
[c][0.2\textwidth]
\begin{itemize}
\legendit{first}{\color{blue}}
\legendit{second}{1.5}{\color{red}}
\legendit{third}{2.25}{\color{green}}
\legendit{fourth}{3}{\color{orange}}
\legendit{fifth}{5}{\color{cyan}}
\end{itemize}
\end{minipage}

```

L'esempio presentato fa uso della libreria `ocgx`, che il pacchetto `ocgx` (ISAMBERT e GABORIT, 2012) mette a disposizione per l'uso diretto con TikZ e consente di rendere visibili o invisibili gli oggetti, creando un bottone da cliccare per farli apparire o scomparire.

- La seconda applicazione consente di avere grafici con coordinate cliccabili. Questo “effetto” è ottenuto mediante l'uso della libreria `clickable` di `pgfplots`, come mostra il seguente semplice esempio:

```

% nel preambolo
\usepackage{pgfplots}
\pgfplotsset{compat=newest}
\usepgfplotslibrary{clickable}
...
\pgfplotsset{/pgfplots/annot/%
point format={(\%.2f, \%.2f)}}
\begin{tikzpicture}
\begin{axis}[]
\addplot[red,domain=-3:3,
samples=201,]{x^2};
\end{axis}
\end{tikzpicture}

```

Questo codice consente di ottenere il grafico di una parabola con coordinate cliccabili: aprendo il file pdf con Adobe Acrobat Reader e cliccando su un qualsiasi punto del grafico, appaiono le coordinate del punto cliccato. È ovviamente possibile personalizzare il comportamento delle finestre pop up che appaiono cliccando e altri aspetti (si veda la figura 2).

Un risultato simile può essere ottenuto inserendo i punti da coordinate date in tabella, come mostra questo esempio, tratto dal manuale di `pgfplots` (FEUERSÄNGER, 2015):

```

% preambolo
\usepackage{pgfplots}
\pgfplotsset{compat=newest}
\pagestyle{empty}
\usepgfplotslibrary{clickable}

```

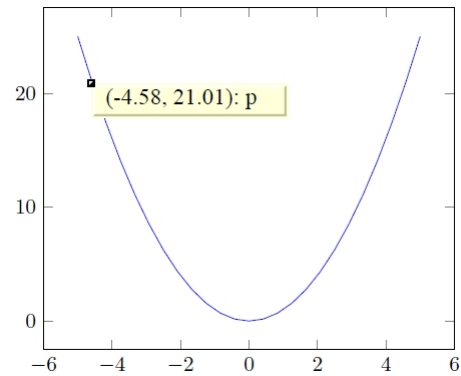


FIGURA 2: Una parabola con coordinate cliccabili

```

...
\begin{tikzpicture}
\begin{axis}[%
clickable coords={(\xy):
\thisrow{label}},%
scatter/classes={%
a={mark=square*,blue},%
b={mark=triangle*,red},%
c={mark=o,draw=black}}]
\addplot[scatter,only marks,%
scatter src=explicit symbolic]%
table[meta=label] {
x      y      label
0.1    0.15    a
0.45   0.27    c
0.02   0.17    a
0.06   0.1      a
0.9    0.5      b
0.5    0.3      c
0.85   0.52    b
0.12   0.05    a
0.73   0.45    b
0.53   0.25    c
0.76   0.5      b
0.55   0.32    c
};
\end{axis}
\end{tikzpicture}

```

Questo codice produce un grafico con punti creati dalle coordinate x e y delle prime due colonne. Il simbolo del punto (cerchio, quadrato, triangolo) è attribuito in base alla lettera che compare nella terza colonna. Le coordinate del grafico sono cliccabili: vengono mostrate sia le coordinate dei punti, che la lettera che consente di attribuire il simbolo.

La libreria `clickable` rende cliccabile l'intera area del grafico e mostra con finestre pop-up le coordinate di ogni punto di quell'area, indipendentemente dal fatto che sia un punto del grafico oppure no. Nella sezione riguardante le applicazioni didattiche (v. par. 4) presenteremo alcune possibili strategie di risoluzione di questo “inconveniente”.

4 Applicazioni didattiche

I pacchetti finora presentati hanno applicazioni molto utili dal punto di vista didattico: spesso gli studenti faticano a “vedere” ciò che si spiega loro e le animazioni, i grafici interattivi e quelli cliccabili possono aiutare a risolvere questi problemi.

Presentiamo qui due applicazioni: le lezioni teoriche (servendoci di pacchetti per le presentazioni, ad esempio beamer) e le esercitazioni on line realizzate con il pacchetto minerva (MESSINEO e VASSALLO, 2013).

4.1 Le lezioni teoriche

In genere, durante una lezione teorica, il supporto delle slide diventa fondamentale quando si vuole illustrare graficamente un certo concetto o una certa proprietà.

Alcuni esempi molto efficaci didatticamente possono ad esempio essere trovati sul sito del prof. Mařík (<http://user.mendelu.cz/marik/latex/>).

Si pensi, ad esempio, di voler illustrare come varia il grafico della funzione $y = (x \pm p)^2$ al variare del parametro p . Presentiamo due soluzioni.

- La prima è realizzata con i pacchetti beamer, pgfplots e animate e prevede la realizzazione di un file pdf di immagini composto da varie pagine (con l’uso del comando \foreach di pgfplots, che dà risultati simili al comando \multido già illustrato), che viene poi “animato” mediante l’uso del comando \animategraphics.

Il seguente codice consente di creare le immagini

```
% nel preambolo
\usepackage{pgfplots}
...
\foreach \p in {0,{white,opacity%
=0},5/green,10/red,15/brown,%
20/orange,25/yellow}
{\begin{tikzpicture}
\begin{axis}[xlabel=$x$,
ylabel=$y$,
ylabel style={rotate=-90},
no marks,
axis x line=center,
axis y line=center,
yticklabels={}]
\addplot[color=black,line width=2pt,
domain=-22:22,opacity=0.2,dashed]%
{x^2}node[right,pos=.1]{$x^2$};
\pgfmathparse{\p+22}%
\let\A=\pgfmathresult
\pgfmathparse{\p-22}%
\let\B=\pgfmathresult
\edef\temp{
\noexpand\addplot[color=\y,
line width=2pt,domain=\B:\A]
{(x-\p)^2}node[right,pos=.1]
{\noexpand\scriptsize{$(x-\p)^2$}}
node[coordinate,pin=above:%
{\noexpand\scriptsize{$V(\p,0)$}}]
at (axis cs: \p,0){};
\temp\end{axis}
```

```
\end{tikzpicture}}
```

L’uso del comando \foreach consente in questo caso di utilizzare un ciclo nel quale ruotano contemporaneamente due variabili: la prima è il parametro p , che in questo caso assume solo i valori di una lista predefinita, la seconda è il colore che il grafico deve assumere in corrispondenza di ogni valore di p .

L’animazione viene prodotta invece con il seguente codice:

```
% preambolo
\usepackage{pgfplots}
\usepackage{animate}
...
\begin{frame}[fragile]%
{Grafico di $y=(x-p)^2$}
\animategraphics[label=parabola,
controls]{1}{parabola}{}{}
\end{frame}
```

- La seconda è, invece, realizzata con l’uso del pacchetto interactiveplot che, come già detto, al momento non può essere utilizzato insieme ai principali pacchetti per le presentazioni (v. par. 3.3.1):

```
% nel preambolo
\usepackage{interactiveplot}
...
\begin{iplotdd}[width=400,
height=400]
\iplot[var={x,-5,5},color=red,
param={p,0,1,-5,5}] {(x-p)^2}
\end{iplotdd}
```

4.2 Le esercitazioni online

Presso alcune scuole superiori delle provincie di Milano e Varese e presso l’Università Cattolica del Sacro Cuore è in atto da quattro anni la sperimentazione di una piattaforma on line di esercizi di autovalutazione di Matematica (Generale e Finanziaria): si tratta della piattaforma “M.In.E.R.Va.” già illustrata durante il convegno GfIr 2012 (MESSINEO e VASSALLO, 2012). Gli studenti dell’Università Cattolica coinvolti nella sperimentazione svolgono online anche una parte della prova d’esame.

Per la realizzazione degli esercizi di autovalutazione, abbiamo realizzato il pacchetto minerva, illustrato durante il Convegno GfIr 2013 (MESSINEO e VASSALLO, 2013).

Presentiamo qui un esempio di come i grafici interattivi possano essere utilizzati per rendere maggiormente comprensibile la modalità di svolgimento di un esercizio.

Si supponga che sia assegnato un grafico e che si chieda di individuare una possibile espressione algebrica per la funzione. Nel file che contiene tutti gli esercizi relativi all’argomento in questione, sarà presente un problema di questo tipo:

```

\newproblem{
\FPsetpar{a}{2}{6}
\FPsetpar{b}{1}{\a}{\a}
\item \PTs{1} Sia  $f(x)$  la funzione
il cui grafico è
\begin{center}
\begin{tikzpicture}
\begin{axis}[xlabel= $x$ , ylabel = $y$ ,
ylabel style={rotate=-90},
axis x line=center,
axis y line=center,ymin=0,]
\addplot[color=green!50!lime,%
domain=-1.5:1.5]{\b+\a*x^2};
\end{axis}
\end{tikzpicture}
\end{center}
una possibile espressione algebrica
di  $f(x)$  è
\begin{answers}*{1}
\bChoices[random]
\Ans0  $f(x)=\b+x^2$  \eAns
\Ans0  $f(x)=\a x^2$  \eAns
\Ans1  $f(x)=\b+\a x^2$  \eAns
\Ans0  $f(x)=\b-\a x^2$  \eAns
\Ans0 nessuna delle altre risposte è
corretta \eAns
\eChoices
\end{answers}
\begin{solution}
Si consideri il seguente grafico.
Si parte dalla funzione elementare
 $y=x^2$  (in blu), si traccia poi il
grafico di  $y=\a x^2$  (in rosso)
e infine il grafico di  $y=\a x^2+\b$ 
(in verde):
\begin{center}
\begin{tikzpicture}
\begin{axis}[xlabel= $x$ ,ylabel = $y$ ,
ylabel style={rotate=-90},
cycle list={blue,red,green,orange,cyan},
axis x line=center,axis y line=center,
ymin=0,ymax=19,xmin=-1.5,xmax=1.5,
smooth]
\plotit{\function}{first}
\plotit{\a*\function}{second}
\plotit{\a*\function+\b}{third}
\end{axis}
\end{tikzpicture}
\begin{itemize}
\legendit{first}{}{blue}
\legendit{second}{\a}{red}
\legendit{third}{\a}{\b}{green!50!lime}
\end{itemize}
\end{center}
Cliccando sui pulsanti con la  $x$  posti
di fianco a ciascuna funzione, è
possibile visualizzarne o no il grafico.
\end{solution}

```

I comandi `\function`, `\button`, `\plotit` e `\legendit` vengono definiti nel file master (usato per la compilazione dell'esercitazione da somministrare agli studenti), in modo del tutto analogo a quanto visto nel primo esempio del paragrafo 3.3.2:

```

\newcommand{\function}{x^2}
\newcommand{\button}[2]{\tikz[base%
\node[fill=#2!30,ogc button=#1]{};]}
\newcommand{\plotit}[2]{\addplot+%
[ocg={name=#2,ref=#2}]{#1};\label{#2}}
\newcommand{\legendit}[4]{\ifx&#3&%
\item[\ref{#1}] #2 $x^2$  \button{#1}{#4}

```

```

\else
\item[\ref{#1}] #2 $x^2$ + $\#3$ 
\button{#1}{#4}
\fi}

```

Il codice presentato produce un file interattivo, nel quale agli studenti è dapprima presentato il grafico della funzione $y = ax^2 + b$, con al posto di a e b due valori numerici scelti a caso tra quelli indicati nel comando `\FPsetpar`. Lo studente sceglie la risposta che ritiene corretta tra quelle proposte, clicca il pulsante “Fine Quiz” e può visualizzare solo allora la risposta corretta e la traccia di soluzione che il docente ha scritto (nell’ambiente `solution`). Nella soluzione, il grafico interattivo permette allo studente di visualizzare i passaggi che portano alla rappresentazione del grafico proposto.

5 Conclusioni

Sia le animazioni che i grafici interattivi sono strumenti indubbiamente efficaci dal punto di vista didattico, perché aiutano gli studenti a visualizzare gli “oggetti” di studio. Per le animazioni L^AT_EX fornisce strumenti efficaci e collaudati ed è compito dell’autore scegliere quello che ritiene più appropriato. Invece, per quanto riguarda i grafici interattivi molto dipende dal codice “esterno” che deve essere utilizzato, in particolare Javascript, che obbliga all’utilizzo di Adobe Acrobat Reader come lettore dei file pdf. In alcuni casi, purtroppo, ci sono problemi di compatibilità con i pacchetti utilizzabili per la creazione di presentazioni e di esercizi. Inoltre l’utilizzo del codice Javascript limita fortemente la possibilità di personalizzazione (caratteri, colori, ecc.) degli oggetti grafici che vengono creati. È nostra intenzione sperimentare ampiamente questi strumenti nel materiale didattico che mettiamo a disposizione degli studenti al fine di valutarne la reale efficacia e tentare di risolvere alcuni dei problemi che abbiamo incontrato, in particolare di compatibilità con il pacchetto minerva.

Riferimenti bibliografici

- BALTIERI, N. e SCANDOLA, M. (2014). «Scrivere report statistici con R e Sweave». *ArsTeXnica*, **18**, pp. 32–38.
- BOCK, R., LINARES, P. e TORO, J. (2014). «Il pacchetto interactiveplot». [CTAN:macros/latex/contrib/interactiveplot](#).
- DE MARCO, A. (2009). «Produrre grafica vettoriale di alta qualità programmando Asymptote». *ArsTeXnica*, **8**, pp. 25–39.
- DE MARCO, A. e GIACOMELLI, R. (2011). «Creare grafici con pgfplots». *ArsTeXnica*, **12**, pp. 9–35.
- DRAKE, D. et al. (2009). «The SageTeX package». [CTAN:macros/latex/contrib/sagetex](#).

- FEUERSÄNGER, C. (2015). «Manual for package pgfplots». [CTAN:macros/latex/contrib/pgfplots](#).
- GRAHN, A. (2012). «Il pacchetto movie15». [CTAN:macros/latex/contrib/movie15](#).
- (2015a). «Il pacchetto animate». [CTAN:macros/latex/contrib/animate](#).
- (2015b). «Il pacchetto media9». [CTAN:macros/latex/contrib/media9](#).
- HUSS, W. e MAŘÍK, R. (2009). «sws2tex». <https://bitbucket.org/whuss/sws2tex/src>.
- ISAMBERT, P. e GABORIT, P. (2012). «The ocgx package (version 0.5)». [CTAN:macros/latex/contrib/ocgx](#).
- MESSINEO, G. e VASSALLO, S. (2012). «Test online di matematica: il progetto M.In.E.R.Va.» *ArsT_EXnica*, **14**, pp. 104–112.
- (2013). «Il pacchetto minerva». *ArsT_EXnica*, **16**, pp. 58–67.
- TANTAU, T. (2013). «The TikZ and PGF Packages». <http://sourceforge.net/projects/pgf>.
- VAN ZANDT, T. (2010). «Documentation for multido.tex, version 1.42: a loop macro for generic T_EX». [CTAN:macros/latex/contrib/multido](#).
- ▷ Grazia Messineo
Università Cattolica Milano – IIS
“Falcone-Righi” Corsico
`grazia dot messineo at unicatt dot it`
- ▷ Salvatore Vassallo
Università Cattolica Milano
`salvatore dot vassallo at unicatt dot it`

Questa rivista è stata stampata
presso Centro Stampa e Riproduzione S.r.l.,
Via di Pietralata, 157 – 00158 Roma,
su carta vellum white 80 g.
Copertina: vellum white 260 g.



Ar_sTeX_nica – Call for Paper

La rivista è aperta al contributo di tutti coloro che vogliano partecipare con un proprio articolo. Questo dovrà essere inviato alla redazione di Ar_sTeX_nica, per essere sottoposto alla valutazione di recensori entro e non oltre il 14 Febbraio 2016. 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 d'esempio (.tex).

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

Di volta in volta verrà fissato, e reso pubblico sulla pagina web <http://www.guitex.org/arstexnica>, 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@guitex.org.

ArsT_EXnica

Rivista italiana di T_EX e L^AT_EX

Numero 20, Ottobre 2015

- 5 Editoriale
Claudio Beccari
- 7 I calcoli matematici in pdftex
Claudio Beccari
- 16 Il pacchetto ECG
Ezio Aimé, Marco Scavino
- 23 Canzonieri online con L^AT_EX
Francesco Endrici, Matteo Conci
- 27 Creare un database bibliografico da un file di testo. Una proposta
Matteo Fadini
- 31 Zbl-build: a GUI interface for Biblalex
Guido Milanese
- 35 MlBibT_EX 1.4: the New Version
Jean-Michel Huppen
- 40 Generare documenti L^AT_EX con diversi linguaggi di programmazione
Roberto Giacomelli, Gianluca Pignalberi
- 74 Ricordi di un cacciatore di spazi spuri
Enrico Gregorio
- 84 Easy T_EX automation with **arara**
Paulo Roberto Massa Cereda
- 90 Animazioni e grafici interattivi in L^AT_EX
Grazia Messineo, Salvatore Vassallo

