Numero 25
Aprile 2018

ArsTEXnica

Rivista italiana di TEX e LATEX

GUIT

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

GUIT – Gruppo Utilizzatori Italiani di TEX

ArsTEXnica è la pubblicazione ufficiale del GUIT

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

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

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

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

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

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

arstexnica@guitex.org.

Nota sul Copyright

Associarsi a GUIT

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

Indirizzi

Gruppo Utilizzatori Italiani di TEX
c/o Università degli Studi di Napoli Federico II
Dipartimento di Ingegneria Industriale
Via Claudio 21
80125 Napoli – Italia
`http://www.guitex.org`
guit@guitex.org

Redazione ArsTEXnica:
`http://www.guitex.org/arstexnica/`
arstexnica@guitex.org

Codice ISSN 1828-2369

Stampata in Italia
Napoli: 15 Aprile 2018

ArsTEXnica

Rivista italiana di TEX e LATEX

Numero 25, Aprile 2018

Gruppo Utilizzatori Italiani di TEX

Editoriale

Claudio Beccari

Questo numero 25 di ArsTEXnica è particolare perché contiene ben due articoli tradotti da TUGboat e tre articoli nuovi: uno racconta un uso insolito di arara, un altro i risultati del concorso della scacchiera e l'ultimo l'archiviazione di documenti LaTeX. Credo che tutti e cinque siano piuttosto interessanti. Ma andiamo per ordine.

L'articolo di Barbara Beeton è stato pubblicato sul numero di TUGboat che costituisce allo stesso tempo gli atti della TUG conference del 2017 e del meeting annuale del gruppo polacco GUST. Strano fascicolo in cui appaiono articoli in inglese e in polacco; mi piace molto questa mescolanza molto interculturale dove si dimostra una volta di più che il TeX Users Group non bada assolutamente a questioni etniche e linguistiche quando riunisce gli "adepti" di TeX.

Barbara Beeton ci parla delle strategie che lei usa per scovare gli errori nei file `.tex` che, in qualità di supporto tecnico della $\mathcal{AMS}$, deve rivedere, editare e aggiustare per portarli al livello di qualità delle pubblicazioni della Società. La lavorazione procede in un modo del tutto particolare perché i documenti che arrivano per la pubblicazione arrivano nelle forme più disparate, persino scritti a mano, ma quando scritti sul calcolatore possono essere il frutto di diversi word processor, oppure possono essere scritti in Plain TeX, LaTeX, pdfLaTeX, XeLaTeX, LuaLaTeX e in ConTeXt. La Società deve provvedere anche alle ristampe, quindi anche l'archiviazione è del tutto particolare perché ogni ristampa deve essere identica alla prima edizione.

Tutto ciò richiede strategie particolari che vengono descritte in dettaglio; al 99% sono applicabili anche da ciascuno di noi.

Jean-Michel Hufflen, nostro collaboratore assiduo, ha pubblicato questo articolo sulle alterazioni musicali negli stessi atti della TUG Conference 2017. Ci parla in sostanza della storia delle notazioni musicali per quel che riguarda quei particolari segni che prendono il nome di alterazioni. Il suo articolo è ricco di immagini raffiguranti dettagli di spartiti dove vengono usati quei segni specifici; essi hanno avuto una storia particolare e sembrano tutt'altro che standardizzati. Alcuni dei software per scrivere musica usati con il sistema TeX e i font che fanno parte delle sue distribuzioni non sembrano sempre in grado di gestire tutti questi segni; il problema è particolarmente sentito nelle edizioni critiche dove è molto importante replicare i segni usati dall'autore originale.

Insieme a Paulo Massa Cereda, altro collaboratore assiduo negli ultimi anni e noto creatore del programma arara, descriviamo come scrivere certe funzioni per quel programma che consentano di svolgere compiti insoliti; questi compiti insoliti riguardano la produzione in automatico, a partire da un solo file sorgente, di documenti su formati di carta diversi ma che, sebbene impaginati diversamente, abbiano lo stesso contenuto; il problema sembra banale, ma all'atto pratico non lo è.

Nel mese di ottobre del 2017 lo staff del GuIt ha lanciato un "concorso" fra i soci e i frequentatori del Forum perché si cimentassero con una macro o un ambiente per disegnare una scacchiera con caselle di due colori, uno bianco o chiaro e l'altro nero o scuro, con gli scacchi scuri sistemati sia in orizzontale sia in verticale alternati con gli scacchi chiari, con il vincolo che la prima casella in basso a sinistra dovesse contenere uno scacco scuro. Il numero delle caselle doveva essere un argomento della macro, in modo che si potessero disegnare scacchiere con un numero ragionevolmente arbitrario di celle. Si poteva usare qualunque programma di composizione basato su LaTeX ma era proibito usare direttamente pacchetti che già facessero questo tipo di disegno o di copiarne le macro interne adattandole alla specifica che il numero di caselle doveva essere fra gli argomenti della macro. Le soluzioni pervenute sarebbero state sottoposte al Consiglio Scientifico sia per accettarle, sia per formare una graduatoria di merito; il migliore avrebbe ottenuto dei "regali" non meglio precisati dal GuIt. Le soluzioni accettate, però, sarebbero state pubblicate in un articolo a più nomi.

La condizione perché questa giuria potesse formulare la graduatoria di merito era che i contributi sottoposti al giudizio fossero almeno tre. I membri dello staff del GuIt e di ArsTEXnica erano esclusi dalla competizione.

Peccato che sia arrivato un solo contributo; allora Roberto Giacomelli e il sottoscritto si sono prestati a "esibirsi" con le loro soluzioni, non per rivaleggiare con l'unico concorrente, ma per mostrare altre soluzioni alternative con soluzioni creative.

I risultati a me sembrano molto interessanti, non perché uno di questi sia un mio frutto, ma perché le soluzioni sono valide tutte e tre e affrontano il problema posto dal concorso con pacchettini che non contengono più di una cinquantina di righe di codice, ma sono decisamente diverse l'una dall'altra: una soluzione minimale ma che rispetta perfettamente le clausole del concorso; un'altra

soluzione è professionale dal punto di vista della qualità della programmazione, ed una terza molto versatile produce grigliati non solo per scacchiere a norma di concorso, ma anche per cruciverba con tutti i comandi necessari per inserire i numeri e gli scacchi del cruciverba.

Emanuele Somma ci propone un articolo per gestire i documenti salvati in certi archivi in modo che sia possibile salvare il sorgente LaTeX con i metadati archivistici nell'archivio e quando un utente vuole vedere quanto archiviato, il corrispondente file PDF, completo dei metadati archivistici, viene creato al volo e reso accessibile all'utente. Questo modo di archiviare i file TeX, invece dei file PDF, o DOC, o DOCX, o in altri formati, permette di archiviare file di solo testo e solo le inclusioni devono essere verificate per l'archiviabilità. Non è un modo di eludere le norme ISO sull'archiviabilità, ma rende l'operazione molto più semplice a spese di un po' di elaborazione da parte del firmware dell'archivio. Anche il caricare nell'archivio nuovi documenti o versioni aggiornate diventa più semplice per gli autori se usano gli script proposti da Somma. L'articolo non riguarda esclusivamente lavori concernenti qualche scienza particolare, ma si riferisce esplicitamente ai documenti composti con LaTeX.

Ringrazio tutti coloro che hanno collaborato per la realizzazione di questo numero di ArsTEXnica, la Redazione, il Consiglio Scientifico, i numerosi revisori editoriali. Sta a voi lettori giudicare se quanto esposto in questo numero di ArsTEXnica è di livello corrispondente alle vostre aspettative.

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

Debugging LaTeX files
*Illegitimi non carborundum**

Barbara Beeton

Sommario

Almeno una volta nella propria attività di utente LaTeX ci si è trovati di fronte a problemi spinosi quando la compilazione si interrompe per qualche oscuro motivo. Si sa abbastanza bene come comportarsi in presenza di problemi semplici, ma ci sono alcune situazioni nelle quali i metodi tradizionali non bastano.

Questo articolo descriverà le strategie e le tattiche per affrontare i problemi emersi durante un lungo periodo della mia attività, come membro del gruppo di sostegno tecnico presso l'American Mathematical Society ($\mathcal{AMS}$), per gestire i problemi degli autori e del personale tecnico. Si incontreranno sia casi semplici sia casi insoliti, con indicazioni utili per tutti al fine di evitare problemi durante il proprio lavoro.

Abstract

Every LaTeX user has, at least once in her career, been faced with a thorny problem when compilation shuts down for some obscure reason. How to deal with simple problems is reasonably well known, but there are situations when the time-honored methods fall short.

This article will present strategies and tactics for dealing with the many types of problems that have arisen during long experience as a member of the $\mathcal{AMS}$ technical support staff, handling questions from authors and the editorial staff. Both common and uncommon glitches will be visited, with a bias toward avoiding problems in one's own work—something for everyone.

*Claudio Beccari ha tradotto Beeton (2017) col permesso dell'autore. Il titolo non è stato tradotto, rimanendo quello originale. La frase in simil-latino è una frase fatta del mondo anglosassone; significa qualcosa come *Non farti demolire dalle avversità*; oppure *È brutto ma tocca farlo lo stesso*. Non ho modificato l'*Abstract* lasciando l'originale. Inoltre devo segnalare che i *bug* del software inglesi corrispondono per similitudine fonetica ai *bachi* italiani; il verbo inglese *to debug* con i suoi derivati non si può tradurre in italiano con una sola parola più significativa di *correggere*; per non appesantire la traduzione ho lasciato quasi sempre la parola inglese. Come traduttore non mi sono limitato a creare la versione italiana, ma a beneficio del lettore italiano mi sono permesso di aggiungere molte mie note che estendono un poco quanto scritto dall'autore di questo articolo; le mie note sono tutte marcate alla fine con (*N.d.T.*). In realtà in italiano esiste il verbo *spulciare* che significa, letteralmente, "mondare dalle pulci". Purtroppo il suo utilizzo nell'accezione di *to debug* è impossibile perché nell'uso corrente significa "dare un'occhiata rapida e distratta". (*N.d.T. e d.R.*)

1 Retroscena

Nel 2016 l'$\mathcal{AMS}$ ha totalizzato la pubblicazione di circa 60 000 pagine di libri e riviste, la maggior parte delle quali da file LaTeX composti e inviati dagli autori. Il fatto che un articolo per una rivista venga accettato è basato sul valore scientifico del contenuto, valutato da un comitato editoriale e da scienziati competenti che si basano su un documento elettronico, non necessariamente composto con un qualche metodo di videoscrittura, ma talvolta anche scritto a mano. Non viene assegnato nessun valore alla presentazione, ma solo al contenuto. I libri vengono acquisiti sotto contratto dal gruppo editoriale; i membri del gruppo e gli autori dei testi conoscono LaTeX ma non sono necessariamente TeXnici esperti. Quello che ci arriva da pubblicare è ciò che dobbiamo affrontare.

Supponiamo che un lavoro accettato sia preparato in LaTeX (se non lo fosse, un tecnico competente provvederebbe a renderlo tale per l'uso successivo); la qualità dei testi inviati è molto varia e offre una ricca serie di opportunità per sperimentare (e migliorare) le proprie capacità di *debugger*.

La produzione si svolge su una rete di sistemi Linux. La libreria disponibile è divisa in tre parti: TeXLive, che viene aggiornato al massimo una volta all'anno; una versione locale di font e di file "pubblici" (che talvolta includono versioni aggiornate di ciò che farà parte della successiva collezione di TeXLive); macro, font ed altri strumenti che sono di proprietà esclusiva dell'$\mathcal{AMS}$. Tutto è archiviato con Subversions, con gli archivi dei libri e degli articoli che risalgono fino a una ventina di anni addietro. Le versioni di (La)TeX e di tutti i pacchetti usati sono registrati insieme ai main file per ciascun lavoro pubblicato usando il pacchetto snapshot, così che se si manifestasse la necessità di rielaborare il documento si potrebbe ricreare l'ambiente di composizione originale. Questo insieme di cose fornisce la stabilità necessaria per produrre continuamente nuove pubblicazioni restando in grado di gestire le riproduzioni, le revisioni e le conversioni delle pubblicazioni esistenti in altri formati, compresi gli ebook.

Come descritto sopra, questo metodo di lavoro è efficace e affidabile una volta che i file che rappresentano il manoscritto sono pronti per es-

sere inviati allo stampatore. Ma prima di questo momento può succedere ogni genere di cose. Un principio guida prevale su ogni altro: se qualcosa va storto, deve essere possibile recuperare prontamente e con assoluta affidabilità un punto stabile da cui riprendere il lavoro.

2 Preparazione — pianificare in anticipo

Esistono delle convenzioni che, seguite con attenzione, alla lunga possono rendere la vita più facile. Per prima cosa bisogna scegliere buoni strumenti acquisendone la completa padronanza.

Lo strumento di gran lunga più importante è un editor o un IDE.[1] L'autore di questo articolo usa `emacs` ma sono disponibili altre scelte, alcune utili per un solo utente su una singola macchina; altre predisposte per il lavoro cooperativo in rete; altre scelte sono una via di mezzo fra questi due tipi. Una lista di tali strumenti si trova in una risposta data nel sito `tex.stackexchange` (di qui in avanti denominato `tex.sx` (StackExchange, 2017)).[2,3]

L'autore preferisce elaborare i file usando la linea di comando. Questo le permette di correggere interattivamente semplici errori (come comandi scritti in modo errato) evitando così ritardi e l'eventualità che si manifestino sequele di errori irrilevanti causati dal comando errato. (Ma non bisogna dimenticare di correggere anche il file sorgente prima della compilazione successiva.).[4]

Tra le funzionalità più utili per il debugging ci sono le seguenti:

- una buona funzionalità nella ricerca di stringhe;
- il controllo dell'appaiamento corretto sia dei vari tipi di parentesi e altri delimitatori, sia delle coppie `\begin`/`\end` relative allo stesso ambiente;
- la possibilità si spostarsi direttamente ad una determinata riga di codice;
- la capacità di contare il numero di istanze dei delimitatori di vario genere.

Un altro fattore importante è la disposizione e il relativo metodo di accesso alle cartelle e ai file. È raccomandabile *evitare gli spazi nei nomi dei file*; infatti non tutti i sistemi operativi gestiscono tali spazi correttamente (alcuni non li gestiscono affatto). Inoltre alcuni sistemi operativi distinguono le lettere maiuscole dalle minuscole — per evitare problemi *si usino sempre solo le lettere minuscole nei nomi dei file*; le cifre e i trattini non danno fastidio nei nomi dei file, ma è bene evitare punti in eccesso (per esempio nomi di file che prima del punto dell'estensione contengono altri punti) e i caratteri che hanno un significato speciale per TEX, per esempio il tratto ribassato '_'; per essere più chiari, un file dal nome `articolo_del 31.05.2016.tex` potrebbe dare molto filo da torcere, mentre se il nome fosse `articolo-del-31-05-2016.tex` non ci sarebbero intoppi di nessun genere.

Se si mantengono i file entro dimensioni ragionevoli alla lunga questo torna molto utile. Per un lavoro importante come un libro, si metta ciascun capitolo in un file a se stante, controllandone la compilazione mediante un main file o un "driver". Questo permette di lavorare su un capitolo alla volta servendosi utilmente della funzionalità di `\includeonly`. Se si devono introdurre figure, disegni, tabelle di grandi dimensioni, metterli in file distinti importati con `\input` rende possibile escluderli con un semplice `%` per commentarne la direttiva di immissione (inserire questi grossi oggetti con `\input` di file separati rende anche facile, all'occorrenza, l'operazione di inserirli in posti diversi del documento).

Infine, quando si preparano questi file da immettere, è solitamente una buona idea quella di terminarli con un comando `\endinput` in una riga a se stante preceduta da una riga vuota; questo elimina i problemi che nascono (in modo non voluto) trasmettendo un file da un sistema a un altro; non mettere *mai* `\end{document}` se non solo alla fine del main file.

Un ultimo suggerimento: si impari subito dove trovare il file `log`, prima che sia necessario esaminarne uno. Alcuni IDE nascondono questo file agli occhi dell'utente; se la compilazione va storta e non si può controllare che cosa sta succedendo leggendo il file `log`, si prospettano tempi molto difficili per scoprire che cosa bisogna correggere.

E ancora:

Non si aggiorni il sistema TEX mentre si sta lavorando un documento importante.

Nuove versioni dei pacchetti potrebbero avere funzionalità nuove e incompatibili con quelle delle precedenti versioni, e i vecchi pacchetti non sarebbero più usabili. Naturalmente se l'hardware decide di "defungere" a quel punto, quanto detto

1. IDE: Integrated Development Environment; ambiente integrato per lo sviluppo del software. Nelle guide del GUIT è spesso denominato all'inglese come *LATEX friendly editor* o come *shell-editor*. (*N.d.T.*)

2. LATEX Editors/IDEs, `http://tex.stackexchange.com/q/339`

3. L'indirizzo web `http://tex.stackexchange.com/q/339` punta ad un filone di discussione un po' datato; esso contiene informazioni interessanti, ma, per esempio, alcuni editor non esistono più o le pagine web loro collegate non funzionano più. Le caratteristiche dei vari editor non sono tutte ugualmente aggiornate; tuttavia si tratta di un ottimo elenco. Nelle guide del GUIT, sono indicati alcuni IDE gratuiti che hanno molte delle caratteristiche indicate in questo articolo. (*N.d.T.*)

4. È anche possibile usare uno *shell editor* adeguatamente configurato (vedi più avanti) per seguire una tecnica simile: al verificarsi di un errore ortografico nel nome di un comando, prima si corregge il file sorgente, così che sia già pronto per la compilazione successiva, poi si interagisce con la console dell'IDE per inserire il comando corretto. (*N.d.T.*)

sopra non è una regola da seguire. Ma certamente è stato fatto un backup completo, non è vero?

3 Isolate e proteggete il vostro collaudo

Bisogna usare solo copie dei file da collaudare.

Se l'errore che si sta cercando di correggere non è semplice come un banale refuso, bisogna proteggersi da possibili disastri creando un ambiente di collaudo speciale. Come minimo si esegua un backup dei file; può andare bene anche un file `zip` dove si sia salvata l'intera cartella di lavoro con le sue sottocartelle, e che sia riposto in un luogo sicuro, magari su un supporto esterno. Si conosce bene la situazione corrente e all'occorrenza bisogna essere capaci di ripristinarla rapidamente e in tutta sicurezza.

Per nessun motivo al mondo si modifichi l'unica copia di qualunque file.

Ancor meglio, se si ha spazio nel disco, si crei un'apposita area di collaudo, identica sotto ogni aspetto importante a quella *vera* di lavoro; si esegua ogni sperimentazione in questa area di collaudo.

Se il lavoro consiste di diversi file, si cominci a copiare *solo* il main file — quello che legge tutti gli altri — nell'area di collaudo; questa sarà la "cavia" su cui lavorare.

Si creino e si usino dei link simbolici per accedere agli altri file. Per i sistemi Linux basta eseguire il seguente comando

`ln -s` ⟨*cartella*⟩

aggiungendo quella ⟨*cartella*⟩ al `PATH`.[5]

Si elabori il documento in modo interattivo. In questo modo errori semplici possono venire corretti al volo prima che producano pletore di messaggi d'errore privi di senso. (Bisogna ricordare sempre di eseguire sia l'immissione interattiva della correzione, sia la correzione nel file che contiene l'errore.) E, se si manifesta un errore non correggibile al volo (come un ambiente ignoto o un ambiente non terminato), la compilazione può essere fermata subito e si può correggere l'errore prima di ricominciare.

Eseguire una compilazione in "nonstop mode" (il modo generalmente predefinito quando si lancia una compilazione attraverso un IDE) vuol dire provocare il verificarsi di una pletora di errori registrati nel file `log` (fino ad un massimo di 100, dopo di che la compilazione si arresta da sola, ma in modo "traumatico", quindi talvolta con un file `log` non completo), ma un singolo errore, che non sia un banale errore ortografico nel nome di un comando, può provocare in cascata una moltitudine di errori, che non si verificherebbero se non si fosse verificato il primo. Lavorare in "errorstop mode" permette invece di fermare subito la compilazione e interagire con la console immettendo uno dei vari comandi che la console accetti: `X`: fermare correttamente la compilazione; `I`; immettere una correzione; `S` proseguire senza arrestarsi, se non per gli errori gravissimi; `Q`: proseguire in silenzio; `H`: chiedere delucidazioni; eccetera.

Più avanti se ne dirà di più nella sezione 8 "Divide et impera".[6]

4 Alcuni strumenti per la diagnosi interattiva

Sono disponibili alcuni comandi diagnostici per inviare messaggi sia al terminale/console sia al file `log`.

`\message{...}` scrive il messaggio sia nel file `log` sia sullo schermo; può essere usato per segnalare quando la compilazione ha raggiunto determinati punti chiave. Per esempio,[7]

```
\message{Ultimo paragrafo,
     pagina \number\thepage}^^J
```

produce nel file `log` un messaggio del tipo:

```
Ultimo paragrafo, pagina 904
```

`\show` scrive il significato corrente di un comando; l'elaborazione viene sospesa per consentire ulteriore interazione scrivendo nella console il codice `I` o `i`, seguito da ulteriori comandi diagnostici. Esempio:[8]

```
\show\LaTeX
```

produce:

```
> \LaTeX= macro:
-> \protect\LaTeX␣.
```

Invece:

```
\show\protect
```

produce:

```
> \protect=\relax.
```

Per inserire ulteriori comandi diagnostici bisogna fare come indicato sopra e mandare in esecuzione il comando premendo il tasto Enter o Invio; premendo nuovamente questo tasto senza avere inserito nulla nella console fa ripartire la compilazione.

`\showthe` riporta invece il *valore* di un comando che identifichi un registro numerico, dimen-

5. L'operazione è identica per Mac OSX; con sintassi leggermente diversa è possibile anche sulle piattaforme Windows recenti. (*N.d.T.*)

6. *Separa e governa*, noto precetto dell'impero romano. (*N.d.T.*)

7. Per la verità `\message` non è interattivo in senso stretto; scrive qualcosa sulla console e nel file `log`, ma non attende nessun intervento da parte dell'utente. Invece `\typeout` permette una vera interattività; si veda (Lamport, 1994). (*N.d.T.*)

8. Il punto finale ha un valore diagnostico importante; nel primo esempio lo spazio che separa la stringa `\LaTeX` dal punto segnala che quello spazio è parte integrante del nome della macro. (*N.d.T.*)

sionale, o altro;[9] agisce esattamente come il comando `\the`. Per il resto si comporta come `\show`.

`\showthe\hfuzz`
produce:
`> 1.0pt.`

Sono disponibili un certo numero di comandi di tracciamento per mostrare i dettagli del flusso di lavoro. (Attenzione: questi comandi di tracciamento possono produrre troppa informazione rispetto a quella che si è disposti a leggere; quindi è bene essere selettivi.) Il risultato del tracciamento è inviato solo al file `log`, a meno che non sia richiesto diversamente. Questi sono i comandi che l'autore usa più frequentemente.

`\tracingoutput` può essere impostato al valore 1 per riportare in forma simbolica il contenuto delle scatole che vengono inviate al file di uscita.

`\tracingcommands` e **`\tracingmacros`** posti al valore 2 riportano i dettagli completi dell'elaborazione eseguita da LaTeX.

`\errorcontextlines = 200` imposta il massimo numero di righe associate ad un singolo messaggio d'errore; il valore preimpostato è 5 ma spesso mostra troppe poche righe per comprendere l'intera operazione.

`\tracingonline` invia il rapporto di tracciamento sia allo schermo sia al file `log`.

I dettagli relativi a questi comandi (e molti altri comandi `\tracing...` della stessa famiglia) si possono trovare nel *TEXbook* (Knuth, 1994) o in *TEX by topic*[10] (Eijhout, 1991).[11]

5 Il file `log` è amico tuo

Il file `log` riporta la registrazione di ogni azione eseguita — quali file e quali font vengono letti, le assegnazioni di scatole e di contatori, e via di questo passo. Più importante, dal punto di vista del debugging, gli errori sono riportati, spesso con un dettaglio faticoso da leggere, ma accompagnati dal numero della riga del file sorgente nella quale il compilatore crede che si trovino.

Ogni volta si controlli nel file `log` il messaggio d'errore, per esempio:

```
! Undefined control sequence.
l.457 \fobx
           {%
```

Gli avvisi (warning) sono segnalati anche loro ma senza l'indicazione della riga:

```
LaTeX Warning; There were undefined references.
```

L'interpretazione di queste informazioni potrebbe risultare una sfida per l'utente, ma questa informazione è la chiave per indirizzare la ricerca. Se il sistema che si sta usando nasconde il file `log` agli occhi dell'utente, si domandi come trovarlo. E non si cancelli mai un file `log` senza prima averlo esaminato.[12]

Non si deve assumere che il numero indicato per la riga dove si è verificato l'errore sia esatto. Per esempio il campo di azione di un tratto di matematica in linea col testo (ciò che è contenuto fra un segno `$` in apertura e un altro segno `$` in chiusura) non può contenere righe vuote; perciò la mancanza di un segno `$` può essere segnalata in corrispondenza di una riga vuota; può darsi che manchi il segno di chiusura oppure può darsi che si sia immessa una riga vuota all'interno del testo matematico.[13]

L'altro errore associato alla mancanza di un dollaro viene indicato con

```
! Missing $ inserted.
```

quando manca il "`$`" di chiusura oppure quando un simbolo matematico è fuori da un ambiente matematico.

Un errore all'interno di una figura, di una tabella, di una equazione di diverse righe, viene di solito segnalato con un numero di riga che corrisponde alla chiusura dell'ambiente, invece che nella riga dove è presente l'errore; ma anche in questi casi la ricerca dell'errore non è difficile poiché di solito lo "scopo" è relativamente breve.

9. Al pari di `\the`, `\showthe` accetta come argomento anche una delle espressioni `\...expr` quali sono definite dalla sintassi estesa dei motori di composizione; per esempio `\showthe\dimexpr\topmargin+1in` mostra il valore vero del margine superiore. In questo testo il valore conservato nel registro dimensionale `\topmargin` vale `-52.36449pt` (negativo) mentre il valore vero vale `19.9055pt` ≈ `7mm`, come il lettore può verificare misurando direttamente sullo stampato. (*N.d.T.*)

10. Con TEXLive questo testo si può leggere anche con il comando `texdoc texbytopic`.

11. Il pacchetto `trace` offre due altri comandi, `\traceon` e `\traceoff` che eseguono un tracciamento abbastanza completo, omettendo però la maggior parte delle centinaia di righe che ogni cambiamento di font comporta e che di solito danno poche informazioni utili. (*N.d.T.*)

12. Di solito non è difficile trovare dove sia il file `log` perché si trova sempre nella cartella dove risiede anche il main file del documento. Usando quell'applicativo dai nomi più disparati per esaminare il contenuto di cartelle e i file (per esempio, `Explorer.exe` su Windows; `Navigator` su alcune macchine Linux; `Finder` sulle macchine Mac) basta controllare il contenuto della cartella del main file. Si faccia attenzione al fatto che diversi sistemi operativi nascondono l'estensione dei file, se non di tutti, almeno di quelli per i quali esiste un preciso programma per agirvi sopra. Si configuri il programma di esplorazione del disco in modo che mostri sempre l'estensione di tutti i file; con il sistema TEX è molto importante disporre di questa informazione. (*N.d.T.*) Sulle macchine *n*x si può usare anche il comando `find` da terminale. (*N.d.R.*)

13. È per questo che con LaTeX sarebbe opportuno delimitare la matematica in linea con i delimitatori `\(` e `\)` invece di usare i dollari; l'informazione dell'errore risulta molto più precisa. (*N.d.T.*)

Un altro caso in cui l'errore viene segnalato lontano dal punto in cui è presente l'errore si manifesta quando un gruppo non è correttamente delimitato, cioè quando si trovano una graffa aperta "`{`" senza la graffa di chiusura, oppure `\bgroup`, `\begingroup`, `\begin{`⟨*ambiente*⟩`}` scompagnati, mancanti, cioè, del corrispondente comando di chiusura. Nel caso di un ambiente con i delimitatori scompagnati il messaggio d'errore potrebbe assumere un aspetto del genere:

```
! LaTeX error: \begin{⟨amb1⟩}
    on input line nnn
        ended by \end{⟨amb2⟩}
```

Questo verrà segnalato appena si incontra la chiusura non corrispondente all'apertura, ma il numero della riga del comando di apertura dovrebbe essere corretto.

Invece un comando di gruppo scompagnato non verrà segnalato fino alla fine del file in cui l'errore si trova, al limite alla fine del processo di composizione. Alla fine di questo processo il messaggio d'errore si presenta su diverse righe:

```
(end occurred inside a group at level m)
### semi simple group (level m)
    entered at line nnn (x)
### bottom level
```

Qui m identifica quanti gruppi aperti restano tali al momento di terminare la compilazione; x indica qual è l'elemento scompagnato: `\begingroup`, oppure "`{`" o `\bgroup`. Anche qui il numero della riga *nnn* dovrebbe essere corretto ma non è indicato il punto in cui si è omesso di chiudere il gruppo.

Altri messaggi d'errore sono riportati nella documentazione di molti pacchetti. La maggior parte di questi messaggi include l'indicazione del numero della riga, e in generale l'indicazione della riga incriminata è abbastanza buono. Spesso questo è sufficiente per localizzare l'errore così che possa venire corretto senza dover procedere con metodi più complessi di ricerca. Una volta trovato e corretto l'errore, se ne verifica l'effettiva correzione, poi la si riporta nel *vero* file di lavoro, e si continua con il lavoro principale, essendo lecito dimenticarsi delle copie fatte e dell'ambiente di collaudo, perché questi hanno svolto il loro compito; l'albero di collaudo può essere mantenuto, ma solo come struttura perché vi si potranno riversare altri file da collaudare con la stessa metodologia.

Ma, ci si potrebbe domandare, quando il lavoro consiste di diversi file come fa uno ad essere sicuro di quale sia il file il cui numero di riga è stato indicato come sede dell'errore? Si veda il prossimo paragrafo.

La lezione importante a questo punto è questa

Non si cancelli il file `log` finché non se ne sia stata estratta ogni possibile anche minima informazione.

È stato suggerito all'autore di conservare il file `log` sotto un altro nome anche più a lungo perché potrebbe essere utile; infatti potrebbe essere necessario eseguire il confronto di due successivi file `log` quando si vuol capire che cosa è cambiato fra due compilazioni successive.

6 E pluribus unum — ma quale?

Quando si ha a che fare con diversi[14] file, quale bisogna esaminare?

Si supponga che l'errore sia stato segnalato per un file di testo, non per un pacchetto.

Quando il file `log` riporta un numero di riga, la prima reazione è quella di guardare il main file. Ma se quel file è lungo solo 95 righe e il numero riportato è 2345, il conto non torna.[15]

Si faccia allora una copia del file `log` e la si lavori all'indietro partendo dal punto in cui è segnalato l'errore sotto esame. Se alcune pagine sono state prodotte e inviate al file di uscita, il numero della pagina (mostrato fa parentesi quadre: per esempio, `[17]`) può indirizzare verso un capitolo, che idealmente dovrebbe essere in un file a se stante. Altrimenti bisogna eliminare da questa copia tutto quanto non serve per lo scopo.

I messaggi relativi alle scatole troppo piene (o troppo vuote) non servono e le rispettive righe possono essere eliminate. Le coppie di parentesi tonde ben appaiate normalmente includono il nome del file che viene aperto in lettura e informazioni varie sulla elaborazione del suo contenuto. Si cerchino allora gruppi completi fra parentesi tonde, come per esempio

```
(C:/tech-support/debug/preface.tex
Preface
[1] [2]
)
```

e si cancelli l'intero gruppo. Quello che rimarrà alla fine è una parentesi tonda aperta seguita dal nome di un file — il nome del file durante la cui elaborazione si è manifestato l'errore. Il numero segnalato dovrebbe riferirsi a quel file.

14. *E pluribus unum* è un motto degli USA; il latino rende in modo conciso il concetto di "Un solo popolo da diversi popoli". (*N.d.T.*)

15. L'autore di questo articolo lavora anche con l'interprete `tex` nella sua forma originale, quella curata da D.E. Knuth e che, tra l'altro, rappresenta il termine di paragone per stabilire se un programma può contenere la stringa `tex` nel suo nome; questo programma non è dotato di molte delle funzionalità diagnostiche dei programmi di compilazione più recenti. Per esempio, sia `latex` sia `pdflatex` sono gestiti dallo stesso interprete, `pdftex`, per il quale si può applicare l'opzione -file-line-error (di solito questa opzione è preimpostata nelle IDE) che produce messaggi diagnostici della forma *file:riga:errore* comune al formato dei messaggi di molti programmi di compilazione. Con quel tipo di diagnostica, quanto scritto in questo paragrafo potrebbe sembrare superfluo ma non lo è: esistono alcune situazioni in cui il tipo di diagnostica descritto non è disponibile, in particolare quando si verificano errori nei pacchetti di servizio caricati nel file di formato. (*N.d.T.*)

Ma cosa succede se il numero di riga è stato segnalato alla fine della compilazione, cioè in una situazione di `level` m? Questo è il punto in cui un ulteriore comando `\end{document}` torna utile.

Si continui a lavorare con i file di collaudo. Non si tocchino i file veri finché non si trova la fonte del problema.

Si lavori sul main file cominciando dal fondo e si inserisca un `\end{document}` fra due direttive `\include`. L'approccio "dicotomico" è utile in questo caso: si cominci a metà. (Se ne dirà di più nel paragrafo 8 "Divide et impera".) Si compili ciò che è rimasto. Se il messaggio `level` m viene ancora emesso, il problema è nei file inclusi prima di `\end{document}`, altrimenti in quelli inclusi dopo. Si commentino le righe che contengono gli `\include` dei file "innocenti" e si sposti `\end{document}` a metà dei rimanenti; si continui così finché non rimane il solo file "colpevole". Naturalmente questo processo è più complicato se $m > 1$, ma il principio è lo stesso.

7 La pulizia del file

Ad un certo punto si è trovato il file dove si ritiene che si trovi l'errore. Può darsi che la posizione dell'errore sia identificata con precisione. Ma può succedere che si abbia ancora solo una vaga idea di dove cercare. Siccome si sta compilando un solo file di collaudo, si elimini la "confusione" togliendo ciò che non serve.

Si modifichi il main file, inserendo nell'argomento del comando `\includeonly` solo il nome del file sospetto. Si commentino anche le direttive che includono parti (irrilevanti) non introdotte mediante i comandi `\include`:

- pacchetti non necessari ai fini del collaudo;
- indici generali e liste di figure e tabelle;
- qualunque cosa legata alla bibliografia;
- comandi `\printindex` per la creazione di uno o più indici analitici.

Si pulisca anche il file sospetto. Non ci si preoccupi di modificare questo file; questa è una copia di collaudo, no? Queste sono le cose che possono essere tolte, sia pure con la dovuta attenzione:

- le righe con un segno di commento `%` all'inizio;
- le righe contenute fra `\begin{comment}` e `\end{comment}` incluse queste due;
- le righe comprese fra `\iffalse` e il corrispondente `\fi` incluse queste due (infatti questo costrutto equivale ad un commento).

Attenzione a che i gruppi siano correttamente delimitati. Questo coinvolge tutte le coppie `\begin` ed `\end` di ciascun ambiente e tutti i vari metodi di raggruppamento. Si controllino questi elementi mediante la funzionalità di conteggio del proprio editor:[16]

- che il numero di graffe aperte `{` sia uguale al numero di graffe chiuse `}` (attenzione al fatto che talvolta viene inserita la stringa commentata `% }` quando una graffa aperta appare solitaria nel codice; perciò si faccia attenzione a questa eventualità);
- il numero di `\begin` sia uguale al numero di `\end`;
- il numero di `\begingroup` sia uguale al numero di `\endgroup`;
- il numero di `\[` sia uguale al numero di `\]`;
- il numero di `$` sia pari come anche il numero di `$$`.[17]

Si compili quello che è rimasto e si controlli via via il file `log`.

Molti problemi nascono dalla mancanza di accoppiamento degli appositi delimitatori; perciò se si è fortunati non c'è bisogno di cercare altro. Supponiamo invece che la causa dell'errore non sia stata ancora trovata.

8 Divide et impera

Quello che si vuole fare è isolare il capoverso, o la più piccola porzione del file, responsabile dell'errore. (Si lavori sempre su copie del file originale; è utile disporre anche di una seconda copia di riserva.)

Si cerchi un buon punto verso la metà del file che costituisca la fine di un capoverso. Vi si inserisca il comando `\endinput` preceduto da una riga vuota. Si faccia attenzione di non inserire tale comando all'interno di un gruppo di qualunque tipo, in particolare che non cada dentro il corpo di un ambiente, cioè fra un `\begin` e il corrispondente `\end`. Si compili questo file ridotto. Se non viene più segnalato nessun errore, il problema risiede nella seconda parte del file (dopo `\endinput`) non ancora elaborata. Si cancelli la parte "sana" del file e si ripeta l'operazione precedente spostando `\endinput` dentro la parte rimasta usando le stesse cautele. Si ripeta questa procedura finché il problema non viene localizzato in un frammento molto piccolo del file che si sta collaudando. Se la soluzione è ovvia, la si corregga e si collaudi il risultato. Si applichi la correzione anche al file di

16. Si potrebbe aggiungere il consiglio di impostare l'editor in modo che usi gli "hard wrap", in modo che ogni riga che compare nella finestra di editing sia numerata per suo conto, e non che i capoversi formino un sola lunga riga che nella finestra di editing compare ripiegata e numerata solo all'inizio del capoverso; questo migliora la diagnostica relativa alla ricerca della riga che contiene l'errore che si sta cercando. (*N.d.T.*)

17. Lavorando con LaTeX, le coppie di dollari non dovrebbero essere mai presenti, perché è considerato un "peccato mortale" delimitare le formule in display con i doppi dollari. (*N.d.T.*)

collaudo intero (di cui si era conservata una seconda copia) e lo si compili; quando si è sicuri di avere corretto bene l'errore, si applichi la correzione al *vero* file del documento e se ne verifichi ancora la correttezza.

Ma se la soluzione non è ovvia?

Se quello che rimane è ancora troppo grande per identificare velocemente il problema — per esempio potrebbe essere una lunga dimostrazione i cui passi sono esposti mediante una lista — si faccia una copia del file con un altro nome e si continui a lavorare sul file di collaudo. (A questo autore è successo varie volte di modificare la propria *copia* che non è quella immessa dal main file. Questo genere di distrazioni può portare all'esasperazione.)

Si riduca ancora il file di collaudo commentando quegli elementi che sembrano innocui, ma per ora non si cancelli nulla — quello che si pensa sia innocuo, potrebbe invece essere parte del problema. Si continui ad iterare questo procedimento finché non c'è più la possibilità di eliminare altro materiale senza eliminare l'errore (non ancora identificato). Quello che è rimasto è diventato un esempio minimo (non-)funzionante, quello che in gergo viene indicato con la sigla "MWE" (Minimum Working Example) [o "EMC" (Esempio Minimo Compilabile) (*N.d.T.*)] — anche se non è ancora funzionante perché contiene ancora il problema non identificato.

Si esamini ciò che è rimasto nel file e

Si faccia attenzione agli indizi contenuti nel file `log`.

Naturalmente se si sa cosa correggere e come farlo lo si può verificare subito eseguendo le opportune modifiche nel file di collaudo e rilanciando LaTeX per confermarlo. Se funziona, si possono applicare quelle modifiche al file *vero*, e *se non si incontrano altri problemi* si è a posto.

Se si incontrano altri problemi si ricomincia da capo, ma questa volta si sa come procedere.

Pero c'è un'area non ancora esplorata, cioè quando un errore si manifesta in qualcosa che si trova prima di

```
\begin{document}
```

Si veda nel paragrafo 10.

Ci si ricordi che ci sono altre tecniche che si possono seguire prima di chiedere aiuto ad altri.

9 Talvolta sono necessarie azioni più drastiche

In questo paragrafo continuiamo a riferirci ad errori che compaiono nel corpo del documento.

Una volta che il problema è stato ridotto alla creazione di un MWE, diventa utile servirsi di altri strumenti diagnostici per ottenere maggiori informazioni. In aggiunta ai comandi mostrati nella pagina 7, nella sezione 4 relativa alla diagnosi interattiva, i seguenti comandi possono risultare molto utili. (Informazioni più dettagliate su questi comandi si possono trovare nel testo *TEX by topic* (Eijhout, 1991).)

- `\tracingmacros` riporta i dettagli sullo sviluppo delle macro e sui valori dei loro argomenti;
- `\showboxdepth` specifica quanti livelli di "inscatolamento" mostrare durante il tracciamento dei comandi eseguiti, dove di solito il numero è uguale a `\maxdimen` (in pratica un numero grandissimo che sta per "tutti");
- `\showboxbreadth` specifica il numero di elementi da mostrare ad ogni livello.

Ne esistono anche altri, ma questi sono generalmente i più utili.

Se si è disperati (e masochisti) si può specificare `\tracingall` ma districarsi fra tutta l'informazione che viene riportata nel file `log` impegna pesantemente la pazienza e la mente dell'utente; di solito si riesce a trovare un approccio "più semplice". Si può vedere nel file `plain.tex` quanti e quali tipi di informazione vengono forniti.

Ma se si è forzati ad eseguire il tracciamento, potrebbe esserci una strada più facile.[18]

10 Nel caso sia necessario un aiuto maggiore

Alcune risorse utili sono disponibili on-line. Potrebbero esserci altre persone che hanno incontrato lo stesso problema.

Gli archivi nel sito di `tex.sx` (StackExchange, 2017) sono un buon punto di partenza.[19] Se non si trova nulla che assomigli al problema che si sta affrontando, si può sempre formulare una nuova domanda d'aiuto. (Bisogna registrarsi nel forum, se non lo si è già fatto in precedenza.) Per ottenere i migliori risultati è molto opportuno accludere un MWE; ma se si è seguita la procedura indicata, se ne dispone già di uno — l'esempio minimo (non) funzionante col quale si è "combattuto" fino al momento di chiedere auto ad altri. Si tolga qualunque commento, e se opportuno (e possibile), lo si renda "anonimo" sostituendo al testo vero un testo di riempimento.[20] Si renda questo MWE il più piccolo possibile purché continui a replicare il problema.

18. Con il pacchetto `trace` e i suoi comandi `\traceon` e `\traceoff` si può attivare un tracciamento molto selettivo; inoltre questi comandi sospendono il tracciamento sia della matematica sia della gestione dei font, riducendo di parecchie migliaia il numero delle righe da leggere. Ciò non toglie che se non si esegue un tracciamento selettivo, si rischia di doversi districare fra decine di migliaia di righe nel file `log`; l'uso intelligente di un IDE permette di ricercare punti particolari nel listato del tracciamento. (*N.d.T.*)

19. Anche il forum del GuIt, `http://www.guitex.org` è particolarmente utile per gli utenti italiani o che conoscono la lingua italiana. (*N.d.T.*)

20. Fra gli altri esistono i pacchetti lipsum e kantlipsum; il primo produce interi capoversi scritti in latino maccheronico, mentre il secondo usa capoversi in inglese che assomigliano

Che cosa appariva sullo schermo:

```
Overfull \hbox (23.1113pt too wide) in paragraph at lines 3288-3301
\OML/cmm/m/it/10.95 a$\T1/ptm/m/n/10.05 , as in
```

Il corrispondente contenuto nel file log:

```
Overfull \hbox (23.1113pt too wide) in paragraph at lines 3288-3301
\OML/cmm/m/it/10.95 a$\T1/ptm/m/n/10.05 , as in Ÿ[], is a degree-
```

Che cosa c'era nel file sorgente:

```
..., as in \S\ref{SS:changing}, is a degree-1 ...
```

Figura 1: Un bel rompicapo

Si accludano anche righe importanti tratte dal file `log` e una spiegazione di quello che si già è fatto per affrontare il problema. I partecipanti al forum sono competenti e amichevoli verso i richiedenti aiuto; di solito sono ben contenti di risolvere il rompicapo che il problema pone — ma necessitano di una quantità di informazione sufficiente che permetta loro di sperimentare; fornire un MWE che essi possano copiare e incollare è un'azione che produce riposte più rapide rispetto a quello che sarebbe necessario tirando ad indovinare.

11 Errori nel preambolo

1. Si faccia una copia del file `log` e lo si apra cercando l'ultimo file aperto.
2. Se questo file non è stato aperto tramite un comando `\usepackage`, si torni indietro fino a quando se ne trova uno.
3. Si ha esperienza con i comandi interni di LaTeX?
4. No. Allora questo è il momento di cercare l'aiuto di un esperto. Si vada su `tex.sx` (Stackexchange, 2017). Se nessuno ha mai riferito di questo problema, si inoltri una domanda di aiuto. Bisogna essere specifici includendo il preambolo e il file `log`.
5. Sì. Allora si cerchi di determinare in che cosa consista il problema. Si controllino i rapporti su `tex.sx` (Stackexchange, 2017). Se il problema non è mai stato segnalato, lo si segnali all'autore del pacchetto.

Questo termina la discussione dei problemi che possono manifestarsi nei *propri* file. Il prossimo paragrafo descrive un problema "tosto" che l'autore ha incontrato e che ha richiesto veramente troppo tempo per essere capito e, alla fine, non era veramente un "problema di LaTeX", sebbene fosse proprio lì che il "mostro" si era manifestato.

a frasi Kantiane di un ipotetico testo *Ideale della ragion pratica*. (*N.d.T.*)

12 Un vero rompicapo

Occasionalmente nemmeno il tracciamento è in grado di indirizzare verso la soluzione del problema.

Due fatti sono importanti.

1. Io vivo e lavoro negli USA. La mia postazione di lavoro è caratterizzata con impostazioni locali (presumibilmente appropriate), cioè con la codifica ASCII.[21]
2. Io compilo usando la linea di comando e non uso le impostazioni `\batchmode` o `\nonstopmode`.[22]

Tempo fa, la compilazione di un file si bloccava sistematicamente prima che il file fosse completamente composto, congelando anche l'operatività del terminale su cui lavoravo. Per riprendere controllo della situazione era necessario aprire un'altra sessione (terminale) per forzare la cessazione del processo di compilazione sul primo terminale. Questo infatti permetteva di inviare il comando da tastiera ctrl-C al terminale bloccato perché ritornasse a mostrare il prompt. L'ultima cosa che appariva sullo schermo era un rapporto parziale relativo ad una scatola troppo piena. Sullo schermo appariva sufficiente testo per localizzare il problema nel file sorgente ... solo che il file sorgente aveva un aspetto perfettamente valido (si veda la figura 1). Il file `log` era presente anche se incompleto.

Dopo aver seguito i passi descritti in precedenza, sono riuscita a ridurre il file di collaudo ad una sola riga di testo che formava un semplicissimo capoverso; se toglievo qualsiasi cosa dall'inizio di questo capoverso l'errore spariva. Il problema sembrava collegato alla scatola troppo piena. Ma a questo punto ho chiesto aiuto a qualcuno più preparato di me sui sistemi informatici.

Dopo aver guardato con attenzione quello che appariva sullo schermo e nel file `log` abbiamo notato quello strano carattere — Ÿ. (Questo carattere Ÿ

21. Nell'Europa continentale tutte le lingue usano segni diacritici di vario genere; la codifica ASCII va bene, in pratica, solo per l'inglese. (*N.d.T.*)

22. Generalmente gli IDE sono impostati con `\scrollmode`; i due modi specificati dall'autore si riferiscono al programma `tex` originale e/o ad una impostazione locale. (*N.d.T.*)

ha la codifica Unicode U+0178, mentre il comando `\S` corrisponde al segno §, che occupa la posizione `"78` nel font `cmsy`.) Siccome sono abituata a lavorare con testi in inglese, e molto raramente maneggio lettere accentate, non sono abituata a vedere caratteri non ASCII, e certamente mai in un file `log` relativo a un documento completamente in inglese.

La soluzione per aggirare il problema che mi è stata indicata consisteva nel mettere la linea seguente

```
LANG="en_US.utf8"
```

in un file chiamato `.i18n` nella mia cartella "home". Questo non risolve completamente il problema — il file continua a bloccare lo schermo, ma la compilazione va a buon fine e posso dare il comando ctrl+C per sbloccare lo schermo e riavere il prompt.

Talvolta quello che uno pensa che sia un baco in LaTeX, non lo è affatto. Si mantenga quindi la mente aperta.[23]

13 Aggiunte successive alla conferenza

Questi che seguono sono problemi facilmente identificabili che capitano spesso, ma la sorgente di ogni problema non è sempre nota. Sembra utile indicare qui come identificarli.

1. L'avviso `Missing character`, per esempio:
   ```
   There is no ; in nullfont!
   ```
   è quasi sempre il risultato di un errore sintattico — la mancanza di un 'punto e virgola' — in un ambiente `tikzpicture`. Altri messaggi che coinvolgono segni di interpunzione e che citano `nullfont` possono essere collegati a qualche espressione di `tikz`. Se il carattere mancante è indicato mediante un paio di segni strani, la cosa potrebbe dipendere da un errore di codifica in ingresso.
2. Analogamente avvisi di questo genere che citano nomi di font devono essere gestiti con una ricerca accurata. Spesso non è indicato il numero della riga nel file `log` ma in questo file compare il numero dell'ultima pagina spedita al file di uscita. Si confronti il file sorgente con la pagina composta e si controlli che cosa manca.

14 Ringraziamenti

Grazie al GUST per aver ospitato la Conferenza TUG'17 a Bachotek insieme al loro incontro annuale; e grazie ai partecipanti le cui domande dopo la mia presentazione mi hanno fornito nuove idee utili e interessanti.

Riferimenti bibliografici

Beeton, B. (2017). «Debugging LaTeX files — Illegitimi non carborundum». *TUGboat*, **38** (2), pp. 159–164.

Eijhout, V. (1991). *TEX by topic*. Addison Wesley Publishing Company (UK). Leggibile anche con `texdoc texbytopic`.

Knuth, D. E. (1994). *The TEXbook*. Addison Wesley Publishing Company, Reading.

Lamport, L. (1994). *LaTeX – User's guide and reference manual*. Addison Wesley Publishing Company, Reading, MA, 2ª edizione.

Stackexchange (2017). «`tex.stackexchange.com`». Internet forum con una grande varietà di domande e riiposte. URL `https://tex.stackexchange.com`. Ultima visita ottobre 2017.

▷ Barbara Beeton
American Mathematical Society
Providence, RI, USA
`bnb at ams dot org`

23. Non ci sono abbastanza dettagli in questo racconto per poter trovare una spiegazione per quel che succedeva all'autore. Tuttavia il problema dipendeva dalle impostazioni del terminale sulla sua macchina Linux, e da caratteri Unicode che la configurazione del terminale non poteva gestire; se l'autore avesse usato un IDE, come usiamo quasi sempre noi in Italia, e avesse usato per l'input del suo file sorgente la codifica utf8 che usiamo abitualmente, il problema non si sarebbe nemmeno posto. La raccomandazione delle guide del GUIT è di usare sempre l'opzione utf8 per l'impostazione dei nostri shell editor e di usare font Type 1 o OpenType a seconda del programma di compilazione che usiamo; sempre, anche se dobbiamo comporre un documento interamente in inglese; è vero che questa lingua non fa uso di accenti, ma nel momento in cui si compone una bibliografia inserendo un'opera scritta da un autore non inglese, la probabilità che il suo nome contenga accenti è molto alta; quindi non si sa mai: usare la codifica utf8 anche quando si compone in inglese è utile. (*N.d.T.*)

Storia delle alterazioni musicali*

Jean-Michel Hufflen

Sommario

I simboli adoperati negli spartiti per modificare leggermente l'altezza di una nota sono conosciuti: il diesis (♯) per innalzarla, il bemolle (♭) per abbassarla e il bequadro (♮) per riportarla alla sua altezza normale. Prima di tutto si dà l'etimologia di questi nomi, poi si mostra che le convenzioni seguite in passato sono molto diverse da quelle attualmente in vigore, specie nel caso delle alterazioni *doppie*. Inoltre, le alterazioni presentano interessanti problemi tipografici perché esistono diverse convenzioni con significati precisi: alterazioni a sinistra della nota (con o senza parentesi) o sopra.

I simboli per le alterazioni adoperati nella musica classica o popolare sono compresi in Unicode, così come lo sono alcuni simboli usati per i microintervalli, per esempio quelli per i quarti di tono. Dal nostro punto di vista, la selezione operata da Unicode è discutibile. Al fine di chiarire la situazione, si mostrano le alterazioni maggiormente adoperate per i microintervalli nella *musique orientale* e nella musica contemporanea. Quest'articolo richiede di saper leggere la musica a un livello appena elementare.

Abstract

Signs used throughout music scores in order to change a note's pitch slightly are well-known: the sharp (♯) to raise it, the flat (♭) to lower it, and the natural (♮) to restore it to its normal pitch. First we give the etymology of these names, then we show that the conventions used in the past are very different from those used nowadays, especially if we consider *double* accidentals. In addition, accidentals present interesting typographic problems because there are several conventions with precise meanings: accidentals left to the note (with or without parentheses) or upwards.

Accidental signs used in classical or popular music are included in Unicode, as have some signs used for micro-intervals, such as quarter tones. From our point of view, the selection made by Unicode is debatable. In order to clarify the situation, we show the accidentals mainly used for microintervals in *musique orientale* and contemporary music. This article requires only basic knowledge in reading music scores.

*Questo articolo, pubblicato con il titolo *History of accidentals in music*, è apparso originariamente in Hufflen (2017) ed è stato tradotto da Tommaso Gordini con il permesso dell'autore. Eventuali errori o fraintendimenti del testo originale sono da attribuirsi al traduttore. (*N.d.R.*)

Introduzione

Negli spartiti, un'*alterazione* è un segno messo di solito prima del simbolo di una nota e che segnala un leggero cambiamento della sua altezza. Consideriamo i tasti di un pianoforte: i tasti *bianchi* vengono indicati dalle lettere comprese nell'intervallo che va da 'A' a 'G',[1] mentre quelli *neri* si raggiungono per mezzo di alterazioni. Questi simboli sono ben noti: il *diesis* (♯) innalza l'altezza di una nota di un semitono,[2] il *bemolle* (♭) l'abbassa del medesimo intervallo e il *bequadro* (♮) la riporta alla sua altezza naturale. In LaTeX, essi vengono prodotti rispettivamente dai comandi `\sharp`, `\flat` e `\natural` in modo matematico.

Da un punto di vista tipografico, la scrittura delle alterazioni negli spartiti obbedisce a regole non sempre conosciute con precisione e che, tra le altre cose, si sono trasformate nel tempo. Per di più, oltre ai tre simboli appena nominati ne esistono molti altri, alcuni dei quali sono compresi in Unicode (2016). Secondo noi, la selezione operata da Unicode è abbastanza discutibile. Quindi, lo scopo di questo articolo è quello di dare una panoramica delle convenzioni relative alle alterazioni. Nella prima sezione discutiamo l'etimologia dei nomi di questi simboli; poi, nella sezione 2, ne diamo le regole tipografiche. La sezione 3 è dedicata ad alcune osservazioni sull'organizzazione di caratteri, software e codifiche. Nella sezione 4 analizziamo le alterazioni per i *microintervalli* – più piccoli dei semitoni – utilizzati nella *musique orientale*[3] e nella musica contemporanea. La lettura di questo articolo richiede di conoscere la musica a un livello appena elementare. I lettori interessati a definizioni precise della terminologia musicale possono

1. Questo nel mondo inglese. Alcuni altri Paesi – tra cui Francia, Spagna, Italia e Russia – adoperano i nomi coniati da Guido d'Arezzo (991 o 992 – dopo il 1033) tratti dai versi di un inno latino dedicato a Giovanni Battista: *ut*, *re*, *mi*, *fa*, *sol*, *la*, per C, D, E, F, G, A. In seguito, nel XVI secolo fu aggiunto il nome *si* per sostituire B e nel XVII secolo *ut* venne rinominato *do*; le origini di questi ultimi due nomi sono controverse.

2. Un *semitono* è l'intervallo tra due note suonato da due tasti adiacenti sul pianoforte, indipendentemente dal loro colore.

3. Questa espressione francese comprende la musica classica andalusa, che comincia nel IX secolo nell'Emirato di Cordoba e annovera musica dei Paesi di Nordafrica, Vicino e Medio Oriente. In inglese, la parola *oriental* è usata più comunemente per riferirsi all'Estremo Oriente, mentre *orientale* in francese rimanda spesso al Vicino o Medio Oriente. Ecco perché adoperiamo l'espressione francese.

consultare Jacobs (1988). In Arnold (1983) e Massin e Massin (1985) si possono trovare ulteriori informazioni sulle questioni storiche, ma in francese. Per aiutare i lettori a collocare storicamente i compositori citati nell'articolo, forniamo loro nel testo o nelle didascalie le relative date di nascita e di morte.[4]

1 Origini

All'inizio del Medioevo si conoscevano tre *modi*. Essi non possono essere considerati come scale 'moderne' comprensive di sette *gradi*, dato che erano basati su *esacordi*, cioè gruppi di sei note. Gli esacordi erano:

- l'*esacordo naturale*[5] (*hexachordum naturale*): do, re, mi, fa, sol, la;
- l'esacordo *molle* (*hexachordum molle*): fa, sol, la, si♭, do, re; e
- l'esacordo *duro* (*hexachordum durum*): sol, la, si, do, re, mi.

Perciò, solo la nota 'si' poteva essere bemollizzata.[6] Il simbolo '♭' deriva da una *b rotonda* (*b rotundum*) – scritta originariamente come '♭' – in relazione con l'esacordo molle; il nome francese *bémol* deriva dal francese medievale *bé mol*[7] per *b molle*. I simboli '♮' e '♯' derivano da una *square b* (*b quadratum*) – scritta originariamente come '♮' – in relazione con l'esacordo duro; il nome francese *bécarre* (♮) deriva dal francese medievale *bé quarré*.[8] I simboli di bequadro e diesis derivano entrambi da questa 'b quadrata', mediante due modi diversi di prolungare i lati del 'quadrato'. Il nome francese *dièse* (per 'diesis') deriva dalla parola latina *diesis*: all'inizio, esso indicava un intervallo di quarto di tono nella musica antica; alla fine dell'Impero romano, venne adoperato per indicare l'intervallo di semitono. Per concludere le questioni etimologiche, sembra che il nome inglese *sharp* (in contrapposizione a *flat*) derivi da 'così alto (in contrapposizione a basso) da essere stonato'.

Torniamo ai simboli di bequadro e diesis. Per molto tempo la differenza tra essi fu poco chiara. Spesso gli spartiti antichi adoperavano i diesis allo scopo di innalzare una nota precedentemente

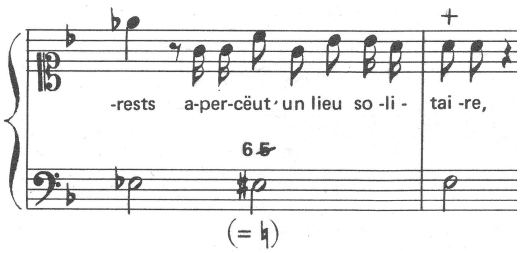


Figura 1: Nicolas Bernier (1664–1734): *Diane*, estratto. (Chailley, 1977, p. 3)

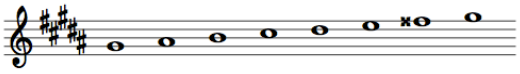


Figura 2: Scala di sol♯ minore.

abbassata. Un simile simbolo di diesis, come mostra la figura 1, nella notazione moderna sarebbe sostituito da un segno di bequadro. Infatti, questi due simboli furono distinti nettamente solo all'inizio dell'età classica. Durante il periodo preclassico, dal xv al xvii secolo, le scale moderne – do maggiore, fa maggiore, sol maggiore, eccetera – stavano emergendo gradualmente. Il primo diesis 'vero e proprio' a essere usato fu il fa♯, poi arrivò il secondo bemolle – mi♭ – e così via: do♯, la♭, sol♯, re♭, re♯, sol♭, la♯, do♭, mi♯, fa♭, si♯. Comparve la nozione di *intervallo enarmonico*: per esempio, la♭ e sol♯ sono la medesima nota, nonostante abbiano nomi differenti.[9]

Le *doppie* alterazioni, per esempio il *doppio diesis* (𝄪) e il *doppio bemolle* (𝄫), furono introdotte nel xvii secolo.[10] Essi innalzano o abbassano l'altezza di una nota di *due* semitoni. All'inizio, l'obiettivo era quello di indicare le *note sensibili*[11] per alcune scale minori: per esempio, fa𝄪 è la sensibile della scala di sol♯ minore, come mostra la figura 2. Il simbolo 𝄪 fu introdotto prima del 𝄫,[12] mentre il ♭ fu coniato prima del ♯.

2 Regole

La maggior parte delle regole che daremo di qui in avanti sono ben note ai musicisti. Le esaminiamo da un punto di vista tipografico.

4. L'articolo originale reca le date anche nella bibliografia. Qui si è deciso di ometterle in quella sede. (*N.d.T.*)

5. Per esempio, l'inno a Giovanni Battista (v. nota 1) è composto in questo modo. Guido d'Arezzo non fu in grado di usare questo brano per dare un nome alla nota 'si', poiché essa non appare nell'esacordo naturale.

6. Inoltre, si fa presente che in quest'epoca i copisti erano incerti se 'si' indicasse si♮ o si♭. L'ambiguità viene rimossa dal sistema di notazione tedesco, in uso ancora oggi: 'B' sta per si♭, 'H' per si♮. Tale sistema è in vigore anche nell'Europa centrale e orientale e in Scandinavia. Ancora, i nomi degli esacordi sono sopravvissuti nel lessico tedesco per indicare i modi delle scale tonali 'classiche': *moll* per 'minore', *Dur* per 'maggiore'.

7. In francese moderno: *bé mou*.

8. In francese moderno: *bé carré*.

9. Infatti, ciò è vero solo per gli strumenti che si basano sul *temperamento equabile* dodecafonico, come per esempio un pianoforte nell'accordatura standard odierna. Affrontare altri temperamenti esula dagli scopi di questo articolo. Inoltre, facciamo notare che tali note enarmoniche non si limitano a quelle suonate mediante i tasti neri di un pianoforte: come controesempio, anche mi♯ e fa♮ sono enarmoniche.

10. (Danhauser, 1929, par. 45) offre un'altra – vecchia – notazione per il doppio diesis: ♯, circondato da quattro puntini. Chi scrive non ha mai visto di persona questo simbolo in nessuno spartito, nemmeno in quelli molto antichi.

11. Una *nota sensibile* si trova appena sotto la tonica di una scala ed è *attratta* da quest'ultima, quindi deve distare da essa un semitono. Nell'armonia classica vengono usati solo modi – maggiore e minore – con note sensibili.

12. Il simbolo 𝄫 non appartiene ad alcuna scala 'classica'; è stato introdotto 'simmetricamente' al 𝄪.

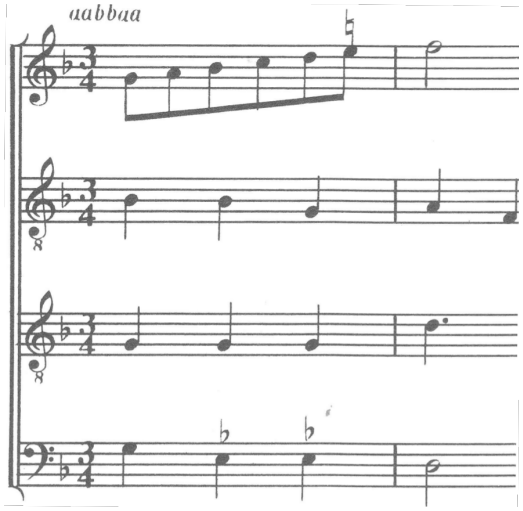


Figura 3: Pierre Attaingnant (1494–1551 o 1552): *Basse dance*, incipit. (Attaingnant, 1993, n. 19)

2.1 Ripristinare le alterazioni

Negli spartiti medievali, le alterazioni erano spesso sottintese. In altre parole, un'alterazione poteva essere omessa ogni volta che risultava ovvia per i musicisti del tempo, i quali erano abituati a ripristinarla.[13] Quando i moderni incisori musicali ripristinano tali alterazioni implicite, devono metterle *sopra* o *sotto* la nota corrispondente – e non a sinistra – come mostra la figura 3. Una simile alterazione ha effetto solo sulla nota cui viene applicata, non anche a quelle successive.[14]

2.2 Ancora sulle doppie alterazioni

Come accennato nel paragrafo 1, a volte negli antichi spartiti un simbolo di diesis aveva un effetto relativo, dato che poteva essere adoperato per innalzare una nota già abbassata. Quest'idea – che oggi sembra strana – è sopravvissuta nel corso delle epoche nell'uso dei simboli doppi 𝄫, 𝄪 e 𝄮 (Chailley e Challan, 1947, par. 82). Per esempio, se una nota doppiamente bemollizzata è seguita dalla stessa nota con un simbolo di bemolle 'semplice', non era corretto scrivere ♭ e basta. Il simbolo 𝄮 significa che il primo ♮ annulla uno dei due semitoni di 𝄫, così ora la nota può essere bemollizzata correttamente. Come mostra la figura 4, lo stesso simbolo ♮ è adoperato qualche volta dopo una nota diesizzata, prima di bemollizzarla. Simmetricamente, l'uso di 𝄪 è analogo a quello di 𝄫. Il simbolo 𝄮 significa che una nota viene riportata alla sua altezza normale dopo un 𝄪 o un 𝄫. Oggi questa regola

13. La cosa può sembrare sorprendente ma, analogamente, oggi molti spartiti di jazz permettono di sottindendere alcuni dettagli: per esempio, spesso i ritmi sono notati in forme semplificate rispetto a ciò che i musicisti suonano veramente. Anche gli accordi vengono sovente semplificati.

14. Di qui la ripetizione del ♭ per due note consecutive nella voce del basso di figura 3. Si noti anche un tipo di *politonalità*, consueta nella musica di questo periodo. Contrariamente a quanto pensano molte persone, politonalità e polimodalità non furono introdotte nel xx secolo.

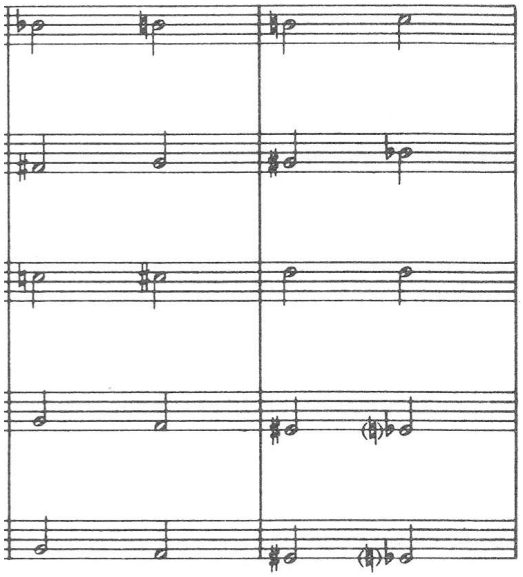


Figura 4: George Gershwin (1898–1937): *Concerto in F for Piano and Orchestra*, i mov., 7 m. dopo 2. (Gershwin, 1987, p. 5)

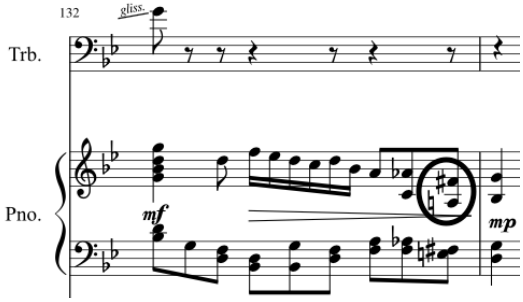


Figura 5: Jean-Michel Hufflen: *Dijon concerto for trombone, string orchestra and piano*, m. 132. (Hufflen, 2015)

complicata – introduce simboli 'composti' davvero inutili – diventa sempre più obsoleta e un'alterazione 'semplice' indica sempre il proprio effetto originale, indipendentemente dalle alterazioni adoperate in precedenza.

2.3 Alterazioni e misure

Nella maggior parte degli spartiti (Chailley e Challan, 1947, par. 79), un simbolo di alterazione agisce sulla nota seguente e su ogni ripetizione di quella nota nella medesima ottava e nella medesima misura, a meno che non venga annullata da un'altra alterazione. Se due note appartenenti a due misure consecutive sono unite da una legatura di valore, l'effetto dell'alterazione dura fino alla fine di questa legatura. Se si adopera un sistema di più righi, un'alterazione usata in uno solo di essi non agisce mai sugli altri. Questa convenzione si è sviluppata gradualmente nel corso del xviii secolo. Prima di tale periodo, le alterazioni si applicavano solo alle note che si ripetevano immediatamente dopo quella alterata o a piccoli gruppi di note, per cui era ovvio che il loro effetto dove-

va continuare.[15] In alcuni spartiti del XIX secolo, le alterazioni vengono applicate alle medesime note nella medesima misura, indipendentemente dall'ottava di appartenenza. Si possono osservare le regole standard nell'esempio di figura 5. Il ♮ evidenziato premesso al la nel rigo della mano destra del pianoforte evita ogni ambiguità, ma è formalmente superfluo: prima di tutto, non appartiene alla stessa ottava del la♭ immediatamente alla sua sinistra e, in secondo luogo, il la♭ della mano sinistra, nella medesima ottava e nella medesima misura, è in un rigo diverso.

Alcuni compositori contemporanei adoperano le alterazioni indipendentemente dai confini di misura, cioè il loro effetto non continua fino al termine della battuta. Le ultime partiture di Hans-Werner Henze (1926–2012) sono esempi in cui le alterazioni vengono applicate solo a una nota o alle note ripetute immediatamente dopo. Come altri esempi – non restrittivi – le alterazioni vengono applicate solo a una nota nelle ultime partiture di Witold Lutosławski (1913–1994) e Henryk M. Górecki (1933–2010). Per segnalare che un'alterazione agisce su una serie di note identiche consecutive, Lutosławski adopera note ripetute prive della testa – come mostra la figura 6 – mentre Górecki ripete esplicitamente l'alterazione prima di ciascuna nota ribattuta.[16]

In caso di dubbio, talvolta compositori ed editori segnano alterazioni supplementari, anche se 'formalmente' inutili. Queste alterazioni sono chiamate *di cortesia* o *cautelative*. Per esempio, se una nota dentro una misura è alterata, l'alterazione di cortesia scritta nella misura successiva permette a chi suona di non confondersi sull'altezza della nota in questione. In particolare, le alterazioni di cortesia andrebbero adoperate quando note unite da lunghe legature di valore si trovano all'inizio di un nuovo sistema. Alterazioni come queste dovrebbero essere messe tra parentesi, ma spesso questa convenzione non viene seguita nella pratica e vengono scritte come alterazioni 'vere e proprie'.[17]

2.4 Armature di chiave

Un'*armatura di chiave* è una serie di diesis o di bemolli associati a una scala. La sistemazione delle alterazioni nelle armature di chiave obbedisce a regole precise sulla loro successione e collocazione sul rigo a seconda della chiave adoperata,[18] ma oggi i software di notazione musicale fanno tutto questo correttamente. Diamo quindi alcuni esempi nella figura 7:

- (a), (c) e (f) usano la ben nota chiave *di violino*;
- (b) usa la chiave di *tenore*, dedicata all'importante serie di strumenti come fagotto, violoncello o trombone;
- (d) usa la chiave di *mezzosoprano* e (e) quella di *soprano*.[19]

In (e) e (f) possiamo osservare che se un'armatura di chiave cambia all'interno di un brano, le alterazioni assenti nella nuova armatura dovrebbero essere annullate da altrettanti ♮, come mostra la figura 7(e, f). Possiamo osservare che oggi questa regola è meno seguita, per cui in un cambio di armatura i ♮ vengono adoperati solo quando la nuova armatura è vuota, come mostra la figura 8.

Ricordiamo che nella musica preclassica alcune armature di chiave sembrano non corrette, specialmente per le scale minori. Per esempio, un brano che si presenta in sol minore adopera l'armatura di re minore, come mostra la figura 3. Più in generale, talvolta le scale minori adoperano un'armatura con un bemolle in meno o con un diesis in più o un'armatura vuota per il re minore. Ulteriori esempi – comprese armature di chiave 'incomplete' per le scale maggiori – si possono trovare in Corelli (1997) e una spiegazione esauriente della questione sta in Fichet (2014). Siamo in grado di chiarire questo *modus operandi* circa le scale minori:[20] esso permette alle scale minori *ascendenti melodiche*[21] di portare il minor numero possibile di alterazioni in armatura di chiave, come mostra la figura 9 per la scala melodica ascendente di sol minore.

3 Intermezzi

Le sezioni precedenti si basano su materiale musicale 'puro'; ora diamo alcuni particolari che hanno a che fare con l'informatica e i software di notazione.

15. Ricordiamo che in quest'epoca alcune alterazioni erano sottintese e applicate dagli interpreti.

16. Facciamo notare che i simboli di ♮ diventano inutili se le convenzioni di far valere le alterazioni solo per una nota vengono osservate negli spartiti privi di armatura di chiave (si veda il paragrafo 2.4).

17. Come abbiamo fatto con la nota circolettata nella figura 5. La partitura di Hufflen (2015) è stata composta con MuseScore (MuseScore, 2017).

18. Armature diverse possono essere adoperate nella *musique orientale* e nella musica popolare, ma questo esula dagli scopi del presente articolo.

19. Oggi queste due chiavi non sono più adoperate con regolarità. La chiave di *soprano* conosce applicazioni scolastiche (negli esercizi di armonia), mentre quella di *mezzosoprano* sopravvive solo nel trasporto. Per dare un'idea circa le relazioni tra di esse, ricordiamo che gli esempi $(a–e)$ cominciano con note alla stessa altezza.

20. La spiegazione di questo *modus operandi* applicato alle scale maggiori richiederebbe un'introduzione ai *modi antichi*, il che non rientra negli scopi di questo articolo.

21. In questa scala, la terza è minore, la sesta e la settima sono maggiori (Chailley e Challan, 1947, par. 151.II). Come illustrato nella figura 9, in una scala minore *discendente melodica* la sesta e la settima sono minori. Nell'armonia classica, queste scale minori melodiche dovrebbero essere adoperate solo per una *melodia*, i successivi accordi che armonizzano una melodia si basano sulla scala minore *armonica*, come descritto nella figura 2 per la scala di sol♯ minore.

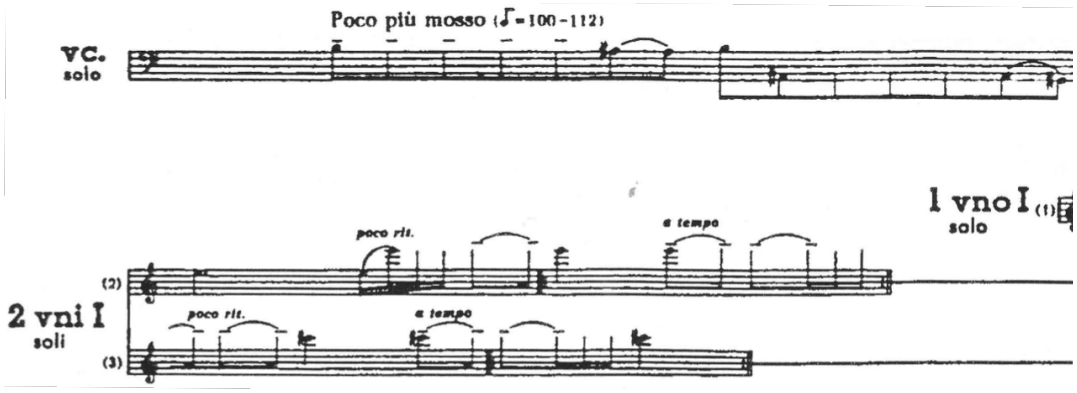


Figura 6: Witold Lutosławski: *Concerto for Cello and Orchestra*: prima di 72. (Lutosławski, 1971, p. 30)

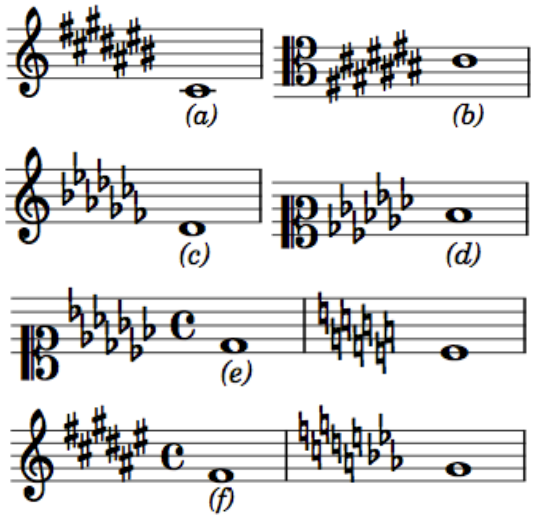


Figura 7: Armature di chiave.

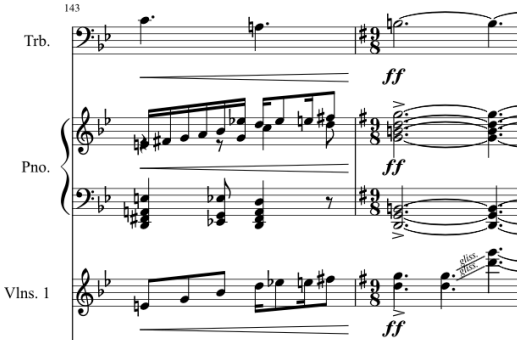


Figura 8: Jean-Michel Hufflen: *Dijon concerto for trombone, string orchestra and piano*, m. 143–144. (Hufflen, 2015)

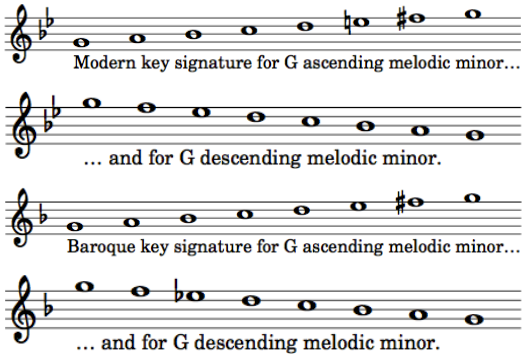


Figura 9: Armature di chiave moderne e barocche per scale minori melodiche.

3.1 '♯' contro '#'

Spesso i caratteri '♯' (U+266F) e '#' (U+0023) vengono confusi. Dal punto di vista grafico sono diversi, dal momento che il secondo è una combinazione di tratti perfettamente orizzontali intersecati da tratti verticali inclinati, mentre il primo si basa su tratti perfettamente verticali intersecati da tratti orizzontali inclinati. Il carattere '#' è il *cancelletto* (o *hashtag*) e si trova sulle tastiere standard, per cui nella pratica rimpiazza spesso '♯'. Un buon esempio di quanto abbiamo appena affermato è il nome del linguaggio di programmazione C# (Microsoft, 2001), scritto con il cancelletto ma pronunciato 'C diesis'.

3.2 Alterazioni in Unicode

In Unicode (2016), le alterazioni 'fondamentali' – ♭, ♮ e ♯ – sono codificate nel *Miscellaneous Symbol Block* (U+266D, U+266E, U+266F). Le altre sono codificate nel *Musical Symbol Block*, dalla posizione U+1D12A alla U+1D133. Le prime due posizioni di questo intervallo sono occupate da 𝄪 e 𝄫.

3.3 Codificare altri simboli musicali

Attualmente Unicode ha mantenuto solo alcuni dei numerosissimi simboli musicali adoperati nel corso delle epoche. Diciamo qui del progetto SMuFL.[22]

22. *S*tandard *Mu*sic *F*ont *L*ayout. Per ulteriori dettagli tecnici, si veda http://smufl.org.

```
\version ”2.18.0”

\score {
 \new Staff {
  \clef ”treble” \time 3/4
  \accidentalStyle Score.default
  r8 bes’8 fes’4. ges’8 | ees’2 f’4 |
  r8 a’8[ d’8. cis’16] g’4 |
 }
 \layout {}
}
```

Figura 10: Esempio di codice LilyPond.

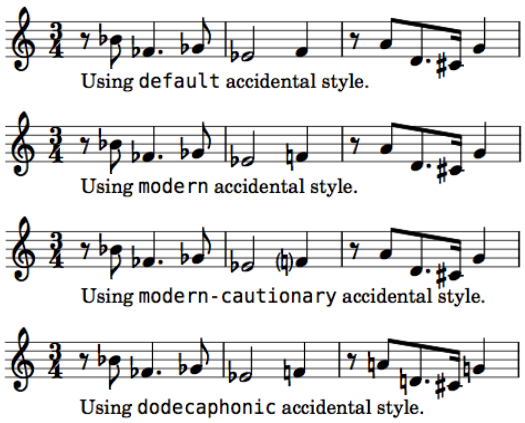


Figura 11: Esempi di stili di alterazione applicati al codice della figura 10.

È una specifica che fornisce un metodo standard per mappare i simboli musicali richiesti dalla notazione musicale convenzionale all'interno della Private Use Area nel Basic Multilingual Plane (U+E000–U+F8FF) di Unicode. In particolare, tutti i simboli introdotti di qui in avanti in questo articolo sono stati mappati. Inoltre, alcuni software musicali – MuseScore (MuseScore, 2017), per esempio – adoperano questa codifica.

3.4 Alterazioni e LilyPond

Per specificare ritmo e altezza di una nota, l'incisore musicale GNU[23] LilyPond (Surhone *et al.*, 2010) fornisce un linguaggio basato su caratteri (in Hufflen (2012) si trovano una breve introduzione al programma e degli esempi). Quando LilyPond compone il sorgente di un brano in uno spartito, adopera *stili di alterazione* per decidere se o come le alterazioni effettivamente appaiono sul rigo. Questi stili – in riferimento alla terminologia di LilyPond possono essere visti come *strategie* – comprendono le seguenti possibilità (LilyPond, 2014).

`default` Le alterazioni vengono inserite o sono lasciate sottintese, secondo la prassi comunemente seguita.

23. Acronimo ricorsivo: *G*nu's *N*ot *U*nix.

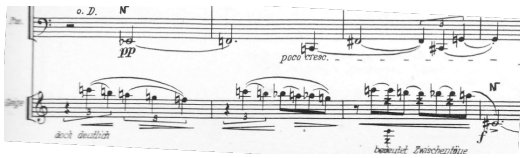


Figura 12: Alban Berg: *Kammerkonzert*, m. 274–277. (Berg, 1925, p. 53)

`modern` Vengono aggiunte alcune alterazioni di cortesia, senza parentesi, per evitare possibili ambiguità.

`neo-modern` Le alterazioni vengono ripetute se la medesima alterazione compare nuovamente nella misura, a meno che la nota alterata in questione non si ripeta immediatamente dopo.

`dodecaphonic` Ogni nota porta un'alterazione, bequadri compresi.

`forget` Le alterazioni non vengono ricordate del tutto.

Gli stili `modern-cautionary`, `neo-modern-cautionary` e `teaching` sono simili rispettivamente a `modern`, `neo-modern` e `dodecaphonic`, tranne per il fatto che le alterazioni supplementari sono tra parentesi, in quanto alterazioni di cortesia. Giusto per portare un esempio brevissimo, la figura 10 mostra il sorgente LilyPond dell'incipit del *Thema* delle *Variationen für Orchester* di Arnold Schönberg (1874–1951) (Schönberg, 1956, p. 7, misure 34–36). L'alterazione premessa a ciascuna nota è indicata con i suffissi '`-is`' per ♯, '`-es`' per ♭ e senza alcun suffisso per ♮. La figura 11 mostra i risultati dopo aver applicato al codice alcuni stili di alterazione di LilyPond. Ci sono altre possibilità per personalizzare l'aspetto delle alterazioni: per esempio, si può permettere l'uso delle alterazioni doppie come 𝄫, 𝄪 e ♯𝄪; si può inserire o rimuovere l'alterazione di una nota legata all'inizio di un nuovo sistema e altre ancora.

Inoltre, accenniamo al fatto che il pacchetto lilyglyphs[24] permette di gestire i glifi musicali disegnati da LilyPond per mezzo di comandi simili a quelli di TEX, purché si adoperino XeLaTeX (Kew, 2007) o LuaLaTeX (Hagen, 2008).[25] Chi scrive ha aggiunto alcuni comandi personali, ma tutti i glifi delle alterazioni 'fondamentali' adoperati in questo articolo sono presi da quel pacchetto.

4 Alterazioni per microintervalli

4.1 Che cosa sono i microintervalli?

I *microintervalli* sono *più piccoli* dei semitoni. All'inizio del XX secolo, i compositori cominciarono

24. È incluso nella distribuzione TEX Live.

25. Tutti questi comandi – comprese le redifinizioni di `\flat`, `\natural` e `\sharp` – vanno dati in modo testuale.

ad adoperarli, in particolare i *quarti* di tono.[26] In quel tempo, tali intervalli venivano indicati per mezzo di notazioni *ad hoc*: per esempio come fece Alban Berg (1885–1935) nel suo *Kammerkonzert* (eseguito per la prima volta nel 1927, figura 12). Un esempio più tardo è fornito dall'ultimo movimento della *Sonata per Violino solo* di Béla Bartók (1881–1945) (Bartók, 1994),[27] composta nel 1944. Alcuni compositori hanno riorganizzato suoni e intervalli in modi del tutto nuovi. Un paio di esempi significativi di importanza storica sono i nomi di Alois Hába (1893–1973) e Ivan A. Wyschnegradsky (1893–1979) (Wyschnegradsky, 1980); entrambi sono andati in questa direzione dopo la Prima guerra mondiale. Inoltre, i quarti di tono sono adoperati in certa musica popolare, specie la *musique orientale*, ma essa non si basa veramente sui quarti di tono nel senso in cui possiamo considerare basata sui semitoni la musica classica (per esempio, nelle scale classiche è il caso dell'intervallo tra si e do). Infatti, la *musique orientale* non ha a che fare con i quarti di tono tra note adiacenti: oltre a toni e semitoni, adopera *grandi toni* (cinque quarti di tono) e *piccoli toni* (tre quarti di tono). Esprimere questa organizzazione per mezzo della nostra notazione occidentale fa sì che compaiano simboli di quarto di tono ma, a rigore, nella *musique orientale* questo intervallo non esiste.

Sono esistite anche altre suddivisioni del tono. Per esempio, Maurice Ohana (1913–1992) l'ha suddiviso in tre parti[28] e Wyschnegradsky in dodici e a volte in più ancora (!) (Wyschnegradsky, 1972). Come accennato nel paragrafo 3.2, alcune alterazioni per i microintervalli appartengono alla codifica Unicode, ma esse non sono in grado di esprimere con precisione gli intervalli più frequenti e più precisi talvolta indicati dai compositori.

4.2 Microintervalli esatti

Esattamente come un semitono è la metà *esatta* di un tono, un quarto di tono è la metà esatta di un semitono, cioè questa divisione ha un quoziente *esatto*.[29] Qualche spiegazione può rendere più chiaro il significato dei simboli di quarto di tono eventualmente adoperati in una partitura. Anche se non esiste una standardizzazione 'ufficiale', quelli adoperati più di frequente sono 𝄲 per il *mezzo diesis*, che innalza la nota di un quarto di tono,[30] e 𝄰 per il *diesis e mezzo*, che la innalza di tre quarti di tono. In particolare, queste notazioni sono adoperate da Iannis Xenakis (1922-2001) (Xenakis, 1969). Esistono notazioni alternative: 𝄯 e 𝄪 rispettivamente.

Il *mezzo bemolle*, che abbassa la nota di un quarto di tono, è spesso indicato con 𝄳; il *bemolle e mezzo*, che l'abbassa di tre quarti di tono, con 𝄫. Le notazioni alternative sono rispettivamente 𝄭 e 𝄬.

Come esempi, nelle figure 13 e 14 possiamo osservare le notazioni adoperate da Alfred Schnittke (1934–1998). Quelle adottate da Krzysztof Penderecki (*1933), Wyschnegradsky e Lutosławski sono mostrate nelle figure 15–17. In queste partiture, i glifi per i mezzi diesis e per i diesis e mezzo sono piuttosto simili. Per quanto riguarda i mezzi bemolli e i bemolli e mezzo, Schnittke e Lutosławski adoperano per i mezzi bemolli glifi 'aperti'. Penderecki usa bemolli completamente neri per i mezzi bemolli e 𝄳 per i bemolli e mezzo. Ulteriori dettagli su queste notazioni e sulle loro varianti si possono

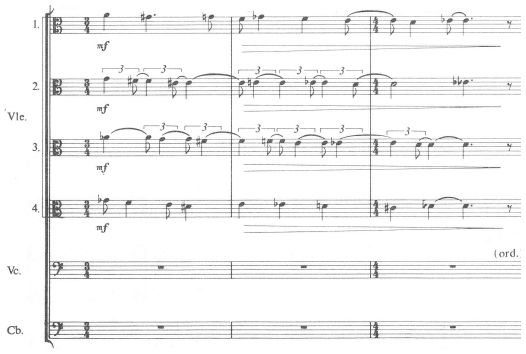


Figura 13: Alfred Schnittke: *Concerto grosso n. 1*, I mov., n. 3. (Schnittke, 1975, p. 2)

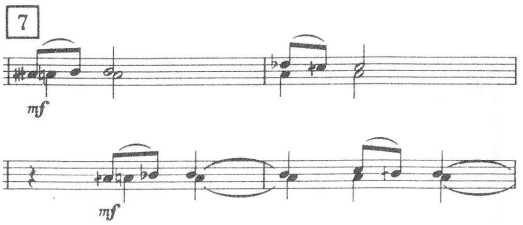


Figura 14: Alfred Schnittke: *Concerto grosso n. 1*, III mov., n. 7. (Schnittke, 1975, p. 42)

26. Chailley (1977) spiega che l'evoluzione della musica attraverso i secoli ha sviluppato accordi che incorporano un numero sempre maggiore di suoni, secondo gli armonici consecutivi in una serie armonica (prima l'ottava, poi la quinta, poi la terza, e così via). In particolare, tale teoria è in grado di spiegare l'introduzione dei microintervalli in questo momento storico (Chailley, 1977, p. 77–79).

27. Questa sonata fu composta per il violinista Yehudi Menuhin (1916–1999). In una lettera del 21 aprile 1944, Bartók scrisse che «gli scalini di quarto di tono possono essere eliminati e sostituiti con versioni alternative». Gli sarebbe piaciuto «ascoltare suonate entrambe le versioni e poi decidere se valeva la pena di usare questi quarti di tono». Sfortunatamente non riuscì mai ad ascoltare questa composizione prima di morire, e spesso al posto degli intervalli originali vengono suonate le loro alternative, mantenute solo nell'edizione di Menuhin.

28. Secondo la sua notazione, l'innalzamento di una nota di un terzo di tono (rispetto a due terzi di tono) è segnalato da '/' (rispetto a '//') a sinistra della nota. I terzi di tono sono stati usati anche nell'ultimo movimento della *Sonata per Violino solo* di Bartók (misure 58–62), nonostante che una versione alternativa, conservata da Menuhin, li elimini (cfr. nota 27).

29. Assumiamo qui il temperamento equabile (cfr. nota 9). In ogni caso, temperamenti non equabili complicano la definizione di quarto di tono, pur portando a risultati precisi.

30. Come accennato nel paragrafo 3.3, tutti i simboli introdotti nella presente sezione sono stati inclusi nella mappatura effettuata come parte del progetto SMuFL. Per esempio, la posizione del 𝄲 è `U+E282`.

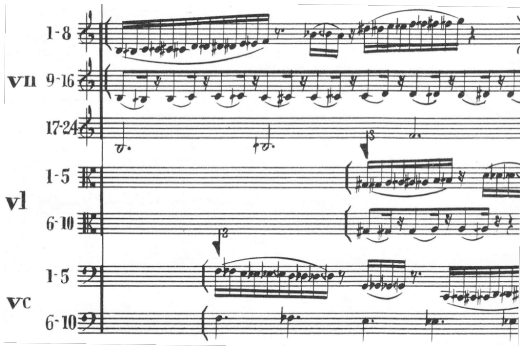


Figura 15: Krzysztof Penderecki: *„Als Jakob erwachte...“*, n. 9. (Penderecki, 1975, p. 10)

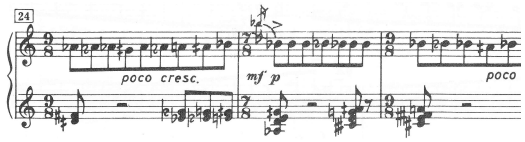


Figura 16: Ivan A. Wyschnegradsky: *24 Préludes im Vierteltonsystem*, Prélude x, m. 24–27. (Wyschnegradsky, 1979, p. 34)

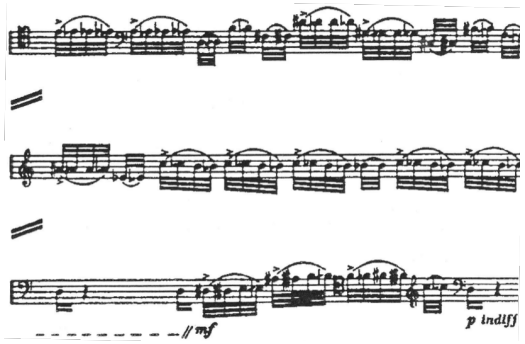


Figura 17: Witold Lutosławski: *Concerto for Cello and Orchestra*, Cadenza. (Lutosławski, 1971, p. 4)

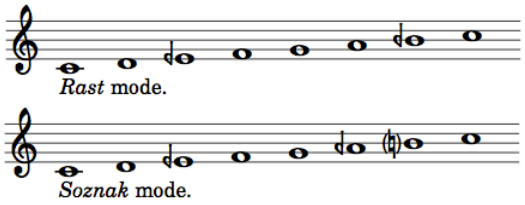


Figura 18: Due esempi di modi nella *musique orientale*.

Tabella 1: Simboli per microintervalli approssimativi.

Diesis	Bemolle	Bequadro
(U+1D130)	(U+1D12C)	(U+1D12E)
⋆(U+1D131)	(U+1D12D)	(U+1D12F)
-	-	-

trovare in Jedrzejewski (2004). Osserviamo che nessuno di questi simboli è stato incluso in Unicode. I glifi definiti da Unicode sono per il momento 𝄲 (U+1D132) e 𝄳 (U+1D133): chi scrive non li ha *mai* visti in *nessuna* partitura.

Concludiamo questa breve ricerca sui quarti di tono offrendo i due esempi di modi nella *musique orientale* mostrati nella figura 18: *rast* e *soznak* (Bohbot, 1983, p. 2) e (Mahdi, 1983, p. 38). Osservando il modo *rast*, possiamo notare un piccolo tono tra II e III grado e un grande tono tra III e IV. Chi è interessato a questi modi, può trovare ulteriori particolari in Bohbot (1983) e Mahdi (1983).

4.3 Microintervalli approssimativi

La tabella 1 elenca i simboli che esprimono altezze *indeterminate* derivati dalle alterazioni classiche (Vogt, 1982, p. 138–139). Per quelli compresi in Unicode diamo anche le corrispondenti posizioni, precedute da ‘⋆’ se il glifo risultante è leggermente diverso.[31] Una freccia verso l’alto (contrapposta a una verso il basso) significa che l’intonazione della nota va leggermente innalzata (e abbassata nel caso contrario). Per esempio, se un interprete suona do𝄲 (mezzo diesis) per do♯, ciò è corretto ma non richiesto; la notazione esprime semplicemente che la nota in questione va messa tra do♮ e do♯. Ancora, essa *dovrebbe* essere più vicina a do♯ che a do♮. Una freccia rivolta in entrambe le direzioni significa ‘nei pressi’ della nota in questione: per esempio, do𝄮 può essere leggermente più acuta o più grave di do♮. Solo con difficoltà siamo in grado di immaginare i risultati di una simile notazione quando a suonare la stessa parte è un gran numero di strumentisti, per esempio tutti i violini di un’orchestra sinfonica. Tuttavia, essa è stata adoperata nella musica da camera, come si vede nella figura 19.

Altre notazioni che esprimono lo stesso comportamento ci arrivano dai *cambi di tono nella voce* del canto bizantino: 𝄬, 𝄭 e 𝄪, come mostra la figura 20.

31. La freccia del carattere Unicode U+1D131 è nell’angolo in basso a destra, mentre quella del simbolo effettivamente stampato si trova nel lato opposto, come mostra la tabella 1.

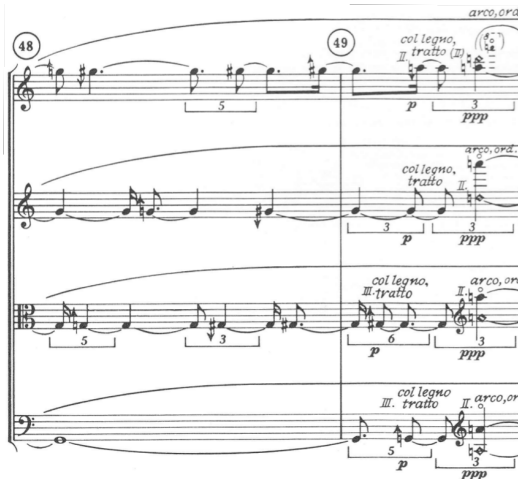


Figura 19: György Ligeti (1923–2006): *Streichquartett Nr. 2*, II mov., m. 48–49. (Ligeti, 1968, p. 16)

5 Conclusioni

Il trattamento delle alterazioni negli spartiti induce all'errore, e la cosa vale in modo particolare per quelli più antichi. I musicologi sono incerti su come interpretarle. Tuttavia, questo sistema è rimasto in vigore per diversi secoli e alcuni tentativi di rimpiazzarlo – per esempio Obouhow (1972) – hanno fallito nell'impresa. Dal punto di vista di software di notazione musicale, a parere di chi scrive il *modus operandi* di LilyPond appare efficace, nel senso che l'aspetto finale di una partitura può essere personalizzato nel dettaglio, con funzioni avanzate assenti in MusiXTEX (Taupin *et al.*, 2002) o in MuseScore (MuseScore, 2017). Per quanto riguarda Unicode, siamo favorevoli all'aggiunta delle alterazioni per i quarti di tono esatti. Naturalmente, Unicode non mira a coprire tutte le nuove notazioni presenti nella musica contemporanea, ma includere questi simboli potrebbe essere interessante per comporre ricerche sulla *musique orientale*. Per quanto ne sappiamo, la maggior parte delle fonti su questo argomento adopera simboli per i quarti di tono esatti e non simboli per i quarti di tono approssimativi come quelli definiti in Unicode.

6 Ringraziamenti

Un ringraziamento particolare va ai traduttori dal polacco: Ryszard Kubiak per il sommario e Jerzy B. Ludwichowski per le parole chiave.[32] Grazie a GUTenberg, il TEX Users Group francese, che mi ha permesso di partecipare al TUG@BachoTEX 2017 offrendomi una sovvenzione. Sono grato anche ai correttori delle bozze di questa versione definitiva: Karl Berry e Barbara Beeton. Da ultimi, ma non per questo meno importanti, Brian Bartling e Gail Berry hanno fornito preziosi consigli su alcune questioni di terminologia musicale.

32. Sommario e parole chiave in polacco, qui omessi, compaiono nella versione originale dell'articolo. (*N.d.T.*)

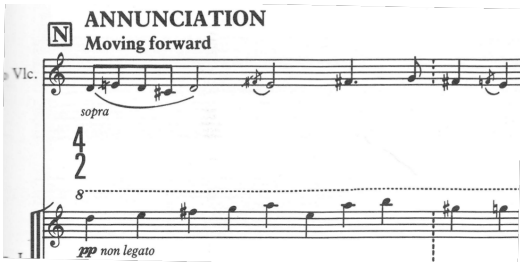


Figura 20: John Tavener (1944–2013): *The Protecting Veil, for Cello and String Orchestra*, II movimento, N. (Tavener, 1993, p. 29)

Riferimenti bibliografici

Arnold, D. (a cura di) (1983). *The New Oxford Companion to Music*. Oxford University Press, Oxford.

Attaingnant, P. (1993). *Danseries à 4 parties (second livre, 1547)*. Éditions Heugel, Paris. Édition par Raymond Meylan.

Bartók, B. (1994). *Sonata for solo violin*. Boosey & Hawkes, London. Urtext edition.

Berg, A. M. J. (1925). *Kammerkonzert*. No. 423. Philharmonia, London.

Bohbot, E. (1983). *Abrégé théorique et pratique de musique orientale traditionelle à 1/4 de ton. Initiation des musiciens occidentaux au 1/4 de ton*. Gérard Billaudot Éditeur, Paris.

Chailley, J. (1977). *Traité historique d'analyse harmonique*. Éditions Musicales Alphonse Leduc, Paris. Nouvelle édition refondue.

Chailley, J. e Challan, H. (1947). *Théorie complete de la musique*. Volume 1. Éditions Musicales Alphonse Leduc, Paris.

Corelli, A. (1997). *Concerti grossi for 2 Violins, Violoncello, Strings and Basso continuo, Op. 6/1–12*, volume 1826–37 di *Eulenburg Miniature Scores*. Edition Eulenburg, Mainz.

Danhauser, A.-L. (1929). *Théorie de la musique*. Éditions Henry Lemoine, Paris. Édition revue et corrigée par Henri Rabaud.

Fichet, L. (2014). *Le langage musical baroque. Éléments et structures*. Musique ouverte. Éditions Minerve, Paris. Édition revue et augmentée.

Gershwin, G. (1987). *Concerto in F for Piano and Orchestra*, volume 1819 di *Eulenburg Miniature Scores*. Edition Eulenburg, Mainz.

Hagen, H. (2008). «The luafication of TEX and ConTEXt». In *BachoTEX 2008 proceedings*. pp. 114–123. April 2008.

Hufflen, J.-M. (2012). «A comparison of MusiXTEX and LilyPond». In *Twenty Years After. BachoTEX 2012 Conference Proceedings*, a cura di T. Przechlewski, K. Berry e J. B. Ludwichowski. Bachotek, pp. 103–108. April 2012.

— (2015). «Dijon concerto for trombone, string orchestra and piano».

— (2017). «History of accidentals in music». *TUGboat*, **38** (2), pp. 147–156. April 2017.

Jacobs, A. (1988). *The New Penguin Dictionary of Music*. Penguin Books, London, 4ª edizione.

Jedrzejewski, F. (2004). *Dictionnaire des musiques microtonoales*. Éditions L'Harmattan, Paris.

Kew, J. (2007). «XeTEX Live». *TUGboat*, **29** (1), pp. 146–150. URL https://tug.org/TUGboat/tb29-1/tb91kew.pdf. April 2007.

Ligeti, G. (1968). *Streichquartett Nr. 2*. ED 6639. B. Schott's Söhne, Mainz.

LilyPond (2014). «Lilypond documentation». URL http://lilypond.org/doc/v2.18/Documentation/web/index.html. March 2014.

Lutosławski, W. (1971). *Concerto for Cello and Orchestra*. PWM Edition, Kraków.

Mahdi, S. E. (1983). *La musique arabe*. Éditions Musicales Alphonse Leduc, Paris.

Massin, J. e Massin, B. (a cura di) (1985). *Histoire de la musique occidentale*. Les éditions Fayard, Paris.

Microsoft (2001). *Microsoft C# Language Specifications*. Microsoft Press, Redmond, WA, USA.

MuseScore (2017). *MuseScore Handbook*. URL https://musescore.org/en/handbook. April 2017.

Obouhow, N. (1972). «L'harmonie totale». *La revue musicale*, (290–291), pp. 25–70. Août 1972.

Penderecki, K. (1975). *„Als Jakob erwachte...“, für Orchester*. Bd. 6639. B. Schott's Söhne, Mainz.

Schnittke, A. G. (1975). *Concerto grosso n. 1 per due violini, clavicembalo [anche pianoforte] e orchestra d'archi*. PH 488. Philharmonia, London.

Schönberg, A. (1956). *Variationen für Orchester, Op. 31*, volume 12 196. Universal Edition, Wien.

Surhone, L. M., Tennoe, M. T. e Henssonow, S. F. (a cura di) (2010). *GNU LilyPond*. VDM Verlag Dr. Müller Aktiengesellschaft & Co. KG, Saarbrücken.

Taupin, D., Mitchell, R. e Egler, A. (2002). *MusiXTEX. Using TEX to write polyphonic or instrumental music*. URL https://www.ctan.org/pkg/musixtex. Version T.104. January 2002.

Tavener, J. (1993). *The Protecting Veil, for Cello and String Orchestra*. Chester Music Ltd., London.

Unicode (2016). «Unicode 9.0.0». URL https://unicode.org. June 2016.

Vogt, H. (1982). *Neue Musik seit 1945*. Philipp Reclam jun. Verlag GmbH, Stuttgart. Dritte, neubearbeitete und erweiterte Auflage. Unter Mitarbeit von Maja Bard, Mathias Bielitz, Hans Peter Haller, Hans-Peter Raiss, Angelus Seipt.

Wyschnegradsky, I. (1972). «L'ultrachromatisme et les espaces non octaviants». *La revue musicale*, (290–291), pp. 71–141. Août 1972.

— (1979). *24 Préludes im Vierteltonsystem Op. 22*. M. P. Belaieff Verlag, Frankfurt am Main.

— (1980). *Manuel d'harmonie à quarts de ton*. Éditions Max Eschig, Paris.

Xenakis, I. (1969). *Nuits. Phonèmes sumériens, assyriens, achéens et autres pour douze voix mixtes solistes ou chœur mixte*. EAS 17045. Éditions Salabert, Paris.

▷ Jean-Michel Hufflen
Université de Franche-Comté
jmhuffle (at) femto-st (dot) fr

Un uso insolito di arara

Claudio Beccari, Paulo Roberto Massa Cereda

Sommario

Si presenta un uso insolito di arara che richiede la definizione di nuove regole con l'uso del linguaggio di espressione `mvel` per Java. L'uso insolito consiste nel chiedere ad arara di impostare certi parametri in linguaggio LaTeX per produrre documenti in formati diversi ma con lo stesso contenuto.

Abstract

We present an uncommon use of arara, that requires the definition of new rules implying the use of the expression language `mvel` for Java. The unusual application consists in requiring arara to set some LaTeX parameters in order to typeset documents in different formats but with the same contents.

Premessa

Va subito premesso che dei due autori, Beccari è quello cha ha avuto bisogno di usare arara in modo insolito. La parte di articolo che si riferisce a questo uso insolito riguarda Beccari.

Il secondo autore ha creato il programma Java arara. In un passato meeting del GUIT Massa Cereda (2015) ha presentato l'ultima versione di questo programma nella sua versione 4.0; però, purtroppo, non è ancora dotato della necessaria documentazione e quindi non fa ancora parte delle distribuzioni del sistema TeX, le quali contengono la versione 3.x adeguatamente documentata.

In questo articolo, quindi, si mostra un uso insolito di arara in versione 3.x. La futura versione 4.0 di arara, già disponibile su `git`, ed entro il prossimo giugno su CTAN, permetterà probabilmente di affrontare l'uso insolito descritto in questo articolo senza bisogno di ricorrere all'uso di `mvel`.

Il secondo autore ha promesso che nella primavera del 2018 metterà finalmente mano alla documentazione e caricherà la versione 4.0 su CTAN.

1 Il problema insolito da affrontare

Beccari scrive documentazione per l'uso di LaTeX mediante i tre programmi pdfLaTeX, XeLaTeX e LuaLaTeX e desidera poterli comporre in tre formati: A4, A5 e B5. Il primo è il formato standard che può essere usato in diversi modi: può venire letto a schermo oppure può essere stampato solo in bianca per poi mettere i fogli in un raccoglitore ad anelli beneficiando, nella volta di ogni foglio, di ampio spazio per le annotazioni. Il formato B5 è più comodo da leggere a schermo, specialmente se si tratta di quello di un laptop. Il formato A5 può essere letto a schermo, ma può essere anche impaginato in segnature dove i fogli A4 sono stampati in bianca e volta, con due pagine su ogni facciata. Se i fogli complessivamente non superano la quindicina, essi possono essere cuciti a quaderno con una pinzatrice "qualunque" e conservati in poco spazio, ma comodi da leggere senza fatica quando se ne avesse bisogno.

La differente larghezza del foglio nei tre formati richiede una impostazione diversa a seconda di quale si sta usando per la composizione, e richiede molta attenzione quando, come succede con le guide, ci sono molti brani di testo, specialmente quando è formato da codice, che devono venire impaginati diversamente. Sono quindi necessari degli switch (delle variabili booleane) che permettano di distinguere sia il formato della carta da specificare fra le opzioni dello statement `\documentclass`, sia le diverse impaginazioni che qua e là sono necessarie.

Per la manutenzione delle guide sopra menzionate è importante che ci sia un solo file da gestire e che le informazioni relative allo specifico file compaiano anche nel nome del file PDF di uscita, e in quelle che possono venire inserite nel frontespizio o comunque usate in altri punti del testo.

Inizialmente Beccari aveva creato tre diversi main file per ciascuno dei tre formati che impostavano il valore delle variabili booleane a cui si è accennato sopra. Tutti e tre importavano lo stesso file contenente il corpo del documento; questo non cambiava da un formato all'altro se non usando gli stati degli switch booleani per aggiustare le impaginazioni generalmente dei formati più piccoli. Ben presto si è accorto che questa operazione era piuttosto delicata e fonte di dimenticanze o di errori, quindi cercava un metodo per potersi liberare da questi problemi. Il risultato è stato il ricorso ad arara.

2 Qualche nozione elementare su arara

La documentazione di arara versione 3.x specifica che ogni esecuzione di arara opera su stringhe che contengono nomi di regole; queste a loro volta possono fare uso di parametri, ma a questi bisogna assegnare un valore prima di invocare la regola.

Le istruzioni per arara vanno messe nel file `.tex` da compilare né più né meno come le righe magiche (commenti di autoconfigurazione). Rispet-

to alle righe magiche, i comandi arara sono più performanti, perché non sarà uno dei tre motori di composizione che agirà sul file .tex direttamente, ma questo file sarà dato in pasto al programma arara, che lo leggerà tutto per estrarre i comandi arara da eseguire. Poi arara stesso lo darà in pasto ai vari programmi che ne faranno l'uso che i comandi arara specificano.

Una serie di comandi arara potrebbe, per esempio, eseguire tutto quello che fa anche lo script latexmk, ma secondo noi arara lo fa molto meglio; nel senso che durante la lavorazione di un documento un certo numero di comandi arara possono venire commentati (come si vedrà più avanti) in modo da limitare le compilazioni allo stretto necessario senza ripetere compilazioni non necessarie, per esempio senza rigenerare la bibliografia se non è necessario. Una delle applicazioni già presenti nella dotazione di regole preconfezionate, contributo di Gregorio (2013), è la gestione del frontespizio delle tesi facendo uso del pacchetto frontespizio.

Nella fattispecie del problema insolito descritto sopra, quello che si chiede ad arara sono i passi seguenti.

1. Ricevere dal comando il valore di un opportuno parametro per scegliere il formato della carta.
2. Generare le opportune istruzioni LaTeX da trasmettere al programma di composizione perché imposti i giusti valori degli switch booleani.
3. Lanciare la composizione con uno dei tre programmi citati all'inizio specificando le opportune opzioni.
4. Solo quando il file .bib della bibliografia è pronto, presumibilmente verso la fine della lavorazione del documento, si può decommentare la riga di arara che lancia l'elaboratore della bibliografia per estrarre i riferimenti citati e formattarli in linguaggio LaTeX per la loro composizione.
5. Dopo aver eseguito il passo precedente arara può lanciare il programma di composizione quelle due o tre volte necessarie perché il tutto finisca con tutti i riferimenti bibliografici corretti e che gli altri riferimenti incrociati siano tutti definiti e nel file .log non compaiano errori.
6. Alla fine, a lavorazione completata, si può attivare l'ultimo comando arara destinato a cambiare il nome del file di uscita PDF, in modo che contenga nel suo nome anche la sigla del formato della carta; in modo più chiaro, se il testo da comporre si chiama guida.tex, il nome guida, viene conservato nelle viscere del programma di compilazione come contenuto della macro \jobname, ma questo nome non può essere cambiato lavorando all'interno di LaTeX. Tocca all'utente, quindi ad arara, copiare il file guida.pdf ottenuto, per esempio, avendo specificato il formato B5, nel file guida-B5.pdf. È meglio copiare che non rinominare il file esistente, perché i collegamenti fra il file .pdf e il file .tex restano attivi fra le due finestre del programma di editing in modo da consentire sia la forward sia l'inverse search così da semplificare la vita dell'utente.

Evidentemente durante la lavorazione del documento non sarà ancora necessario comporre anche la bibliografia con le relative citazioni, per cui gli ultimi tre passi potranno essere indicati nel file sorgente .tex in modo che non vengano eseguiti; in questo modo, anche con documenti semplici, si risparmia molto tempo.

3 Impostare gli switcth logici per trasmetterli al file .tex

È poco noto che l'affermazione presente in quasi tutte le guide di LaTeX, secondo cui non si può scrivere nessun comando prima di \documentclass, è una raccomandazione e non un divieto assoluto. Tant'è che persino il manuale originale di Lamport (1994), il creatore di LaTeX, raccomanda di usare l'ambiente filecontents (con o senza asterisco) prima di \documentclass. Ecco: questo è il punto cruciale della procedura descritta in questo articolo. Basta iniziare il file sorgente .tex con queste righe:

```
\newif\ifacinque \acinquefalse
\newif\ifbcinque \bcinquefalse
\newif\ifaquattro \aquattrofalse
\input{\jobname.cfg}

\documentclass[\ifacinque
  a5paper\else\ifbcinque
    b5paper\else
      a4paper\fi\fi
  ,⟨altre opzioni⟩]{⟨classe⟩}
\ProvidesFile{guida.tex}[2018/02/08
  v.1.1.7 Guida breve]
```

Come si vede, prima di \documentclass compaiono alcune definizioni di switch booleani, impostati tutti tre al valore false e l'inserimento di un file di configurazione con lo stesso nome del main file, grazie all'uso di \jobname, nel quale si troverà l'informazione di quale switch, unico fra i tre definiti, viene posto uguale a true. Sarà compito di arara scrivere tale file con il valore specificato nell'apposito comando arara. Quelle quattro righe fanno tutta la "magia" necessaria per ottenere dal solo file guida.tex un file guida.pdf ed eventualmente, grazie ad arara, i tre file guida-A4.pdf, guida-A5.pdf e guida-B5.pdf composte su carta virtuale dei tre formati diversi.

Va però da sé che la geometria della pagina, e le strutture diverse da usare per i tre formati devono essere assoggettati agli stati dei tre switch non diversamente da come si è fatto per scegliere l'opzione di formato da inserire fra le opzioni del comando `\documentclass`.

4 I comandi arara

Quello che bisogna fare ora è mettere le righe magiche per arara, in teoria in qualunque parte del file sorgente, ma è decisamente meglio inserirle all'inizio, dopo le righe magiche di autoconfigurazione dell'editor. Queste righe magiche contengono i comandi arara con i loro argomenti; devono essere scritte ciascuna su una sola riga; qui le spezzeremo per la ristretta giustezza della colonna, ma si deve intendere che le righe che iniziano con un rientro siano la prosecuzione della riga precedente.

```
% !TEX TS-program = arara
% !TEX encoding = UTF-8 Unicode
% arara: writeconfig: { suffix: B5 }
% !arara: pdflatex
% !arara: bibtex
% !arara: pdflatex
% !arara: pdflatex
% arara: pdflatex: { synctex: true }
% arara: rename
```

Alcuni commenti sono necessari.

1. La prima riga non specifica come programma uno di quelli del sistema TEX, ma specifica di usare `arara`, come motore per gestire la composizione; in sostanza si dice all'editor che deve gestire il file da comporre con `arara`, non con pdfLATEX o XeLATEX o LuaLATEX. In questo modo `arara` legge l'intero file, ne estrae le informazioni che concernono i suoi comandi, e quando li ha letti tutti, li esegue nell'ordine in cui sono scritti.
2. La seconda riga, come al solito, dice all'editor di gestire il file sorgente con la codifica utf8, oggi quella da preferire sempre. Non riguarda i comandi arara, ma fa parte delle righe magiche di autoconfigurazione dell'editor, righe che non possono essere messe dovunque, ma solo fra le prime 20 del file.
3. Successivamente compaiono 6 righe di comandi arara; come si vede tutte cominciano con la parola chiave `arara`; se questa parola chiave è preceduta da un punto esclamativo, la riga viene totalmente ignorata dal programma `arara`. I punti esclamativi potranno essere tutti tolti quando si vorranno eseguire anche i comandi che si sono "commentati" agli effetti di arara. Si noti che l'ultimo contiene anche l'opzione per la "sincronizzazione" del file sorgente con il file composto, in modo che viene conservata la possibilità che l'editor possa eseguire la forward e inverse search.
4. I nomi delle regole sono autoesplicativi, ma le regole `writeconfig` e `rename` non corrispondono a nessuna fra le varie decine di regole già predefinite e descritte nella documentazione di `arara`; quindi bisogna definirle apposta. Si noti che mentre `rename` apparentemente non ha bisogno di argomenti, `writeconfig` deve sapere cosa scrivere nel file di configurazione, ed è ciò che appare dopo i due punti fra parentesi graffe; il qualificatore `suffix` servirà per i comandi arara affinché ne facciano l'uso appropriato; si vedrà più avanti meglio quando si descriveranno le due regole `writeconfig` e `rename`.
5. Non è immediatamente chiaro, invece, dove i codici di queste regole debbano venire memorizzati. La documentazione di arara lo spiega in modo abbastanza chiaro; esse vanno conservate una cartella residente nella propria "home"; questo concetto di "home" è chiarissimo con i sistemi operativi Linux (cartella indicata con `~/`) e per il sistema operativo Max OS X (cartella `~/Library/`); per le piattaforme Windows il concetto è chiaro solo per le versioni più recenti, dove per "home" si intende la cartella `C:\Users\`⟨*nome dell'utente*⟩; per le altre versioni si consulti bene la documentazione di arara. Chi scrive sta lavorando su un Mac, quindi ha creato la cartella `~/Library/myararararules`; egli ha messo anche un piccolo file con estensione `.yaml`, nella cartella `~/` dove si indica il path completo della cartella `myararararules`. Di questo si parlerà con maggiore dettaglio nel prossimo paragrafo
6. Va da sé che nel momento in cui `arara` legge il main file, prende nota automaticamente del suo nome e lo memorizza in una sua variabile interna.

Sembra complicato? No, è più lungo da scrivere che da fare.

5 Le regole per arara

Veniamo pertanto ai dettagli dei file per arara. Come si è detto, le regole vanno conservate in una cartella con qualunque nome, ma che sia individuabile con precisione da `arara`. Nella "home" `~/` si è messo il file

`araraconfig.yaml`

il cui contenuto è il seguente:

```
!config
paths:
- /Users/claudio/Library/myararararules
```

Il fatto che contenga la parola `paths` al plurale lascia intendere che si possono usare più percorsi per conservare i file delle regole; in generale è piuttosto prudente agire così perché si possano

```
identifier: writeconfig
name: WriteConfig
command: <arara> @{
  config1 = new java.io.File(getBasename(file) + '.cfg');
  config2 = new java.io.File('suffix.cfg');

  dict = ['A4':'aquattro', 'A5':'acinque', 'B5':'bcinque' ];

  text = '\\' + dict[suffix] + true;

  java.nio.file.Files.write(config1.toPath(), text.getBytes());
  java.nio.file.Files.write(config2.toPath(), suffix.getBytes());

  return isWindows('cmd /c echo', 'true') }
arguments:
- identifier: suffix
  flag: '@{parameters.suffix}'
  default: 'A4'
```

Codice 1: Codice di `writeconfig.yaml`

```
identifier: rename
name: Rename
command: <arara> @{
  config = new java.io.File('suffix.cfg');

  suffix = new java.lang.String(java.nio.file.Files.readAllBytes(config.toPath()));
  suffix = suffix.replaceAll('\n', '');

  cp = isWindows('cmd /c copy', 'cp');
  bn = getBasename(file);

  return cp + ' "' + bn + '.pdf" "' + bn + "-" + suffix + '.pdf"' }
arguments: []
```

Codice 2: codice della regola `rename.yaml`

conservare regole diverse per far svolgere ad `arara` compiti diversi.

In accordo al contenuto di `araraconfig.yaml`, viene creata la cartella `myarararules` in cui verranno inserite le due regole seguenti: `writeconfig.yaml` e `rename.yaml`. Queste, scritte da Paulo Massa, presentano il contenuto descritto nelle due prossime sezioni.

5.1 La regola `writeconfig.yaml`

Il file `writeconfig.yaml` contiene quanto mostrato nel codice 1. Le parole chiave `identifier` e `name` servono rispettivamente per definire il nome della regola, e la stringa da scrivere durante la sua esecuzione; dalla riga 3 fino alla riga 14 si trova la definizione del comando da eseguire; le righe da 15 a 18 specificano gli argomenti e il valore di default di questi argomenti; in particolare, se al comando non si danno argomenti, si assume che si stia componendo il documento su carta in formato A4.

Si noti che vengono definiti due stream di scrittura; uno per scrivere il file di impostazione con lo stesso nome del file sorgente, che `arara` recupera mediante la sua funzione `getBasename(file)`, e l'atro per specificare il suffisso che l'altra regola `rename` userà per aggiungerlo al nome del file PDF finale.

Perciò agendo con `arara` sul file `guida.tex` con la serie di comandi non "commentati" indicati nel paragrafo 4, viene specificato il valore `B5` alla regola `writeconfig` e questa crea il file di configurazione `guida.cfg`, che contiene solo l'impostazione `\bcinquetrue`, e un secondo file `suffix.cfg` che contiene solo il suffisso `B5`; quando il documento finale `guida.pdf` verrà "rinominato" con la regola `rename` esso verrà in realtà copiato nel file `guida-B5.pdf`.

5.2 La regola `rename.yaml`

L'atra regola `rename.yaml`, infatti, contiene quanto riportato nel codice 2; si vede che sfrutta alcune delle cose impostate con la regola `writeconfig`. Infatti recupera il suffisso precedentemente memorizzato nel file `suffix.cfg` e lo usa per generare

il nome del file di destinazione; si vede, infatti che quanto fornito da questa regola è un comando per il sistema operativo (Mac o Linux)

```
cp guida.pdf guida-B5,pdf
```

tanto per continuare ad usare l'esempio descritto nei paragrafi precedenti.

Il codice 2 contiene un test per sapere se si sta componendo su una piattaforma Windows; in questo caso invece di `cp`, che va bene per Mac e Linux, la regola imposta correttamente il comando `copy`. Quindi la regola riportata nel codice 2 può essere usata indifferentemente su tutte le piattaforme più comuni.

6 Commenti

Nel caso specifico, che ha dato origine a questo uso insolito di **arara**, il modo di procedere è il seguente: si inseriscono le righe indicate nel paragrafo 4 in testa al main file del documento da comporre; durante la lavorazione si lasciano commentate le righe con i comandi **arara** preceduti dal punto esclamativo; quando il documento è pronto, si tolgono i punti esclamativi e si esegue l'intera compilazione compresa la bibliografia e le due successive compilazioni in modo da sistemare tutte le citazioni dei riferimenti bibliografici in modo corretto. Si ripetono le stesse operazioni con e senza i punti esclamativi con un diverso valore del formato della carta; nel fare questo si aggiustano in base ai valori degli switch booleani relativi al formato quelle parti dell'impaginazione che presentano dei difetti causati dalla minore giustezza della gabbia del testo. Volendo si potrebbero aggiungere ai comandi **arara** indicati nel paragrafo 4 gli stessi comandi con gli altri valori dei formati della carta, in modo che con un'unica compilazione complessiva del documento in tutti e tre i formati si ottengano i file PDF finali coerenti gli uni con gli altri, nonostante le differenti impaginazioni; chi scrive, tuttavia, preferisce agire su ciascun formato in modo indipendente e dopo ogni correzione del corpo del documento preferisce compilare separatamente i tre file richiesti controllandoli visualmente in ogni dettaglio per essere sicuro che i tre file siano tutti e tre impaginati correttamente.

L'esempio che si è presentato è molto particolare, ma getta una luce sulle infinite possibilità di uso di **arara** per altri casi particolari e insoliti non ancora previsti nella distribuzione del pacchetto **arara**. Fra pochi mesi la distribuzione di questo pacchetto potrebbe essere aggiornata alla versione 4.0, insieme alla sua documentazione; probabilmente per la scrittura di altre regole non sarà necessario ricorrere direttamente ai comandi Java usati nell'esempio riportato in questo articolo; per lo meno dovrebbe essere molto più semplice. Nel frattempo il lettore interessato può farsi venire il desiderio di approfondire **arara** leggendo l'articolo (Massa Cereda, 2015).

Riferimenti bibliografici

Gregorio, E. (2013). «Introduzione a **arara**». PDF document. URL http://profs.scienze.univr.it/~gregorio/introarara.pdf.

Lamport, L. (1994). *LaTeX. A Document Preparation System*. Addison-Wesley.

Massa Cereda, P. R. (2015). «Easy TeX automation with **arara**». *ArsTEXnica*, (20). URL https://www.guitex.org/home/images/ArsTeXnica/AT020/cereda.pdf.

▷ Claudio Beccari
claudio dot beccari at gmail dot com

▷ Paulo Roberto Massa Cereda
Università di San Paolo, Brasile
paulo dot cereda at usp dot br

Il concorso della scacchiera

Claudio Beccari, Roberto Giacomelli, Maurizio Molinaro

Sommario

Questo articolo descrive tre soluzioni al problema della scacchiera. Ciascuna è prodotta da uno dei tre autori e le soluzioni sono molto diverse l'una dall'altra. È molto interessante notare il diverso approccio che ogni autore ha usato per affrontare il problema.

Abstract

This article describes three solutions to the problem of designing a chess board. Each solution is created by a different author and it is very interesting to note the different approach they had in facing the problem.

1 La sfida

Nell'ottobre 2017 lo staff del GUIT ha lanciato una specie di concorso fra gli utenti di TEX. Esso consisteva nel presentare una soluzione al seguente semplice problema:

> Creare il software necessario per disegnare una scacchiera con un numero specificato di righe e colonne che abbia una casella nera nel quadretto in basso a sinistra e che l'intera scacchiera sia poi riempita alternativamente di celle bianche e nere alternate.
>
> Si può usare qualunque dei tre programmi di composizione basati su LATEX: pdfLATEX, XELATEX e LuaLATEX; si possono usare tutti i pacchetti del sistema TEX tranne quelli che già disegnano cose simili e ovviamente non si possono "copiare" le macro di questi pacchetti.

In teoria questo gioco doveva costituire un concorso e le varie soluzioni avrebbero dovuto essere sottoposte al giudizio del Consiglio Scientifico che avrebbe decretato la migliore di queste soluzioni.

Questo guanto di sfida non è stato raccolto da un numero sufficiente di utenti di TEX: la sfida avrebbe avuto il giudizio del Consiglio Scientifico solo se i concorrenti fossero stati almeno tre.

Tuttavia alcuni membri dello staff del GUIT si sono cimentati a loro volta, se non altro per controllare che il problema non fosse tanto difficile da scoraggiare i possibili concorrenti. Le loro soluzioni li hanno convinti che il problema era sufficientemente semplice e hanno lanciato il guanto.

Forse il problema era troppo facile? Forse. Tuttavia, senza mettere a confronto di merito le varie soluzioni, qui se ne descrivono tre:

- una semplice soluzione di Maurizio Molinaro, molto lineare, basata sul pacchetto TikZ;
- una soluzione di Claudio Beccari, molto estesa nelle prestazioni, ma basata solo sulle funzionalità grafiche dell'ambiente picture descritto da Lamport (1994) nella versione del 1994 del mark-up LATEX;
- una soluzione di Roberto Giacomelli, noto ai lettori di ArsTEXnica per la capacità di usare creativamente LuaLATEX e il linguaggio Lua.

Il bando della sfida non specificava se i colori della scacchiera dovessero essere limitati al bianco e al nero; infatti i tre autori hanno impostato le loro soluzioni in modo da consentire l'uso anche di altri colori.

Il bando non specificava nemmeno che i numeri delle righe e delle colonne dovessero essere uguali tanto da costruire una scacchiera di $N \times N$, invece che di $N \times M$ righe e colonne. O meglio, lasciava intendere, anche mediante l'uso della parola "scacchiera", che dovesse trattarsi di un oggetto quadrato. Infatti la soluzione quadrata è stata presunta da tutti e tre gli autori, ma Beccari ha previsto un'opzione per una scacchiera rettangolare e Giacomelli ha previsto che i numeri di righe e colonne vengano sempre specificati entrambi anche se sono uguali.

Il bando non escludeva l'uso di nessun pacchetto, tranne, ovviamente, quelli già esistenti anche se non adattati al numero arbitrario di righe e colonne, come per esempio Fischer (2014); o meglio: questo pacchetto è previsto per il gioco degli scacchi, ma è in grado di rappresentare anche zone parziali rettangolari della scacchiera.

Tutte e tre le soluzioni sono originali; il lettore può esaminarle con attenzione e magari scoprire metodi di programmazione che potrebbero servirgli anche in altre circostanze.

In questo articolo i comandi e/o gli ambienti descritti saranno sempre chiamati col nome "scacchiera". In realtà i pacchettini che contengono i tre codici sono stati leggermente modificati per distinguerli l'uno dall'altro; si ritiene infatti che i lettore interessato possa tranquillamente capire la differenza d'uso che ne viene fatto nei paragrafi che seguono.

```
\ProvidesPackage{ScacchieraMM}[2018-02-05 1.2 Comando per generare una scacchiera]

\usepackage{framed}
\usepackage{ifthen}
\usepackage{tikz}

% Definizione degli indici per i cicli
\newcounter{k1}
\newcounter{k2}

\newcommand{\scacchieraMM}[2]{%
\ifthenelse{#1 < 2 \or #1 > 20}
{\textit{La dimensione richiesta è #1,\\
 e invece dev'essere compresa tra 2 e 20!}}
{\ifthenelse{\isodd{#1}}%
% d dispari
{\setcounter{k1}{\numexpr (#1-1)/2}
 \setcounter{k2}{\numexpr (#1-3)/2}}
% d pari
{\setcounter{k1}{\numexpr (#1-2)/2}
 \setcounter{k2}{\numexpr (#1-2)/2}}

\begin{tikzpicture}[chiara/.style={white}, scura/.style={black!20!white}, scale=#2]
% caselle chiare
\fill[chiara] rectangle (#1,#1);
% primo ciclo
\foreach \m in {0, ..., \value{k1}}
\foreach \n in {0, ..., \value{k1}}
 \fill[scura] (2*\n,2*\m) rectangle +(1,1);
% secondo ciclo
\foreach \m in {0, ..., \value{k2}}
\foreach \n in {0, ..., \value{k2}}
 \fill[scura] (2*\n+1,2*\m+1) rectangle +(1,1);
% contorno
\draw (0,0) rectangle (#1,#1);
\end{tikzpicture}}
}
```

Codice 1: Codice della soluzione di Molinaro

2 La soluzione di Molinaro

Molinaro è l'unico utente di LaTeX, non facente parte dello staff del GUIT, che ha raccolto il guanto di sfida.

Questa è una soluzione estremamente lineare basata sul pacchetto TikZ; Molinaro definisce il comando `\scacchiera` che accetta in entrata il ⟨*numero*⟩ di righe e la ⟨*dimensione*⟩ del lato di ogni cella. Si assume che la scacchiera sia quadrata ed entrambi i comandi obbligatori, quindi la sintassi è la seguente:

```
\scacchieraMM{⟨numero⟩}{⟨dimensione⟩}
```

Se si vogliono le caselle di colore diverso dal nero, bisogna ridefinire le opzioni `chiara` e `scura` presenti nel comando di apertura dell'ambiente tikzpicture; d'altra parte non sarebbe difficile definire due colori, diciamo `colorechiaro` e `colorescuro`, e in quelle opzioni del comando di apertura definire le suddette opzioni in termini di questi colori.

Il codice 1 comincia con la gestione dei limiti accettabili per il numero delle caselle delle righe e colonne; se il numero è inferiore a 2 o superiore a 20, viene emesso un messaggio d'errore e non viene eseguito il disegno.

Se il numero esce dai limiti predetti si ottiene un messaggio d'errore:

> *La dimensione richiesta è 1,*
> *e invece dev'essere compresa tra 2 e 20!*

In vista di due cicli `for` per inserire le caselle chiare e scure nella tabella, vengono caricati due contatori LaTeX, *k1* e *k2*, con cui contare il numero di ripetizioni delle inclusioni delle celle colorate di scuro sopra un background già caricato di celle colorate di chiaro.

Finalmente vengono eseguiti i vari cicli `for` e le caselle vengono messe in posizione.

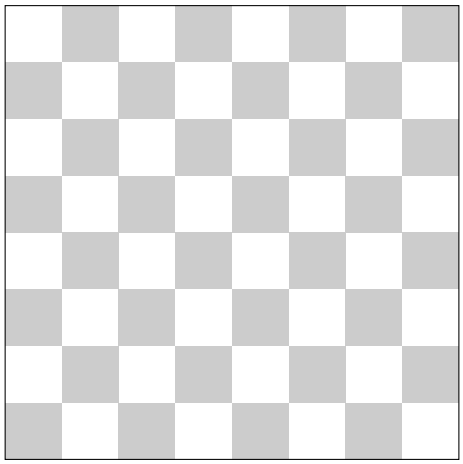
Complessivamente il file di estensione con i comandi appena descritti contiene quanto appare nel codice 1 nella pagina 30.

Alcuni esempi compaiono nella figura 2.

3 La soluzione di Beccari

Beccari è un utente convinto della bontà dell'ambiente picture e non usa spesso TikZ, pur riconoscendone le infinite funzionalità. Egli ritiene che il semplice ambiente picture del nucleo di LaTeX sia altrettanto valido per certi lavori particolari, e quello della scacchiera è uno di questi. Egli usa abitualmente le estensioni documentate da Lamport (1994) nel lontano 1994, diventate effettive quasi 10 anni dopo con la pubblicazione del pacchetto pict2e (Gäßlein *et al.*, 2016) e a cui egli stesso diede qualche contributo.

Scacchiera 8x8 con lato 0,75 cm



Scacchiera 9x9 con lato 0,5 cm

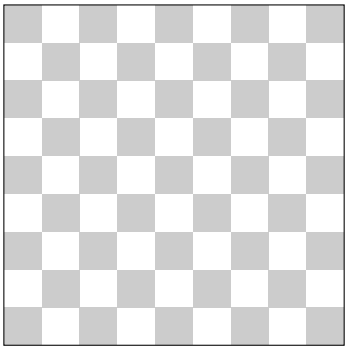


Figura 2: Due scacchiere con numeri di righe

Egli ha anche sperimentato le funzionalità dei pacchetti etoolbox (Lehman e Wright, 2017) e xparse (The LaTeX3 project, 2017); il primo offre molte funzionalità interessantissime per chi scrive macro; il secondo mette a disposizione dei comandi potentissimi per definire nuovi comandi e nuovi ambienti che accettano una moltitudine di argomenti diversi, delimitati e non delimitati; obbligatori o facoltativi, e anche semplici token, il più comune dei quali è l'asterisco (già usato dal nucleo di LaTeX per alcuni ambienti e per alcuni comandi). Egli usa anche abitualmente le estensioni del linguaggio nativo di TeX descritte nella documentazione di etex (The $\mathcal{NTS}$ Team, 1998) e che da diversi anni sono incorporate in tutti gli interpreti del sistema TeX.

Perciò il suo pacchetto scacchieraCB è dipendente da ε-TeX, ma non c'è bisogno di invocare nulla perché le sue funzionalità, benché poco usate dagli utenti del sistema TeX, fanno già parte dei suoi motori di composizione; il pacchetto scacchieraCB dipende da pict2e, etoolbox e xparse che vanno caricati *prima*. Utile, se si vogliono usare le "color expressions", è il pacchetto xcolor, il cui ordine di caricamento non è rilevante.

Beccari ha messo insieme questi tre pacchetti e l'ambiente picture per definire alcune macro che permettono di costruire sia scacchiere sia cruciver-

ba con una moltitudine di varianti sfruttando le potenti funzionalità, specialmente quelle di xparse. Egli ha calcolato che la sua macro di base potrebbe accettare argomenti facoltativi e obbligatori in 128 modi diversi. Il lettore non si spaventi, però: di questi 128 modi diversi in pratica se ne usano una dozzina, ma quel numero la dice lunga sulla potenza delle funzionalità dei comandi definiti con xparse.

Chi scrive si scusa per questa lunga premessa, ma siccome le scacchiere di Beccari e i suoi cruciverba sono molto variegati, era necessario spiegare come e perché.

In sostanza, Beccari definisce un comando `\scacchierainterna` sul quale si basano gli altri comandi e ambienti: il comando `\ComandoScacchiera` e gli ambienti scacchiera e cruciverba. Questi sono poi affiancati da alcune macro che servono per collocare delle cose particolari in celle scelte della scacchiera o del cruciverba: `\GenBox`, `\ScacBox`, `\NumBox`, oltre al comando `\setfontsize` per impostare in modo più semplice il corpo dei caratteri senza dover perder tempo a calcolarne a mano lo scartamento. Egli ha definito anche i comandi `\usecs` e `\whilenum`: il primo molto simile ad un comando di etoolbox; il secondo per evitare di usare il comando interno del nucleo di LaTeX `\@whilenum` che, contenendo il carattere `@`, non è usabile direttamente dall'utente se copia il codice 4 nella pagina 35 nel suo preambolo (invece nei file `.sty` il carattere `@` è considerato una lettera ed è perfettamente lecito).

3.1 La scacchiera di base

La scacchiera di base è formata dallo sfondo, anche colorato di un colore diverso dal bianco, con il reticolo dei quadretti. Dentro questa scacchiera di N righe e M colonne possono apparire caselle scure, di colore anche diverso dal nero, alternati alle caselle chiare (come specificato dal bando del concorso), ma possono apparire anche caselle scure in posizioni "random", scelte dall'utente senza una regola particolare. Nel caso dei cruciverba, caso particolare di scacchiere, possono apparire anche i numerini delle definizioni orizzontali e verticali.

La particolarità della scacchiera di base con le caselle scure prescritte dal concorso è l'algoritmo con cui vengono collocate; esse infatti si trovano solo nelle caselle diagonali dirette da nord ovest a sud est. Tenuto conto delle coordinate di ciascuna di queste caselle, si constata che la somma dei loro indici (ascissa e ordinata) è sempre un numero pari (si ricordi che l'origine degli assi coincide con il vertice in basso a sinistra della scacchiera complessiva, e le coordinate di ciascuna casella sono quelle del suo vertice in basso a sinistra). In questo modo, scorrendo con due cicli gli indici delle righe e delle colonne, ci vuole una casella scura quando la loro somma è pari. Nel codice 4 della pagina 35 si riconoscono infatti le quattro righe che svolgono questa operazione:

```
\I=0\J=0\relax
\whilenum{\N > \I}{{\whilenum{\M > \J}{\unless
\ifodd\numexpr\I+\J \put(\I,\J){\nero}\fi
\J=\numexpr\J+1}}\I=\numexpr\I+1}}%
```

Si notino i comandi `\unless` e `\numexpr` di ε-TeX; il primo rovescia l'esito di un test; il secondo esegue i calcoli nei registri della CPU del calcolatore, invece di usare i registri interni del programma di composizione. Forse in questo caso sono eccessivi [1], ma in generale è meglio ricorrere a questi comandi di ε-TeX, piuttosto che ai comandi nativi o a quelli messi a disposizione dal pacchetto calc (Thorup *et al.*, 2014), raccomandazione valevole anche per i calcoli sulle lunghezze rigide ed elastiche.

Nel codice 4 si nota anche l'uso dei comandi di base dell'interprete del linguaggio LaTeX: `\countdef` e `\dimendef` usati con numeri di registri numerici e dimensionali maggiori di 255. Questi registri sono usabili grazie a ε-TeX; assegnare loro un nome simbolico evita di commettere errori nella programmazione, ma è importante però invocare tali operazioni dentro un gruppo o un ambiente, cosicché alla chiusura del gruppo o dell'ambiente quei registri vengano ripristinati ai valori che essi avevano prima della loro apertura. Ciò evita spiacevoli interferenze con l'uso di quei registri da parte di altre funzioni del programma che si sta eseguendo. A questo scopo servono anche i comandi `\bgroup` ed `\egroup`, che rappresentano delle graffe implicite più visibili delle graffe esplicite ma usabili anche in situazioni insolite[2].

Disponendo di font con i disegni dei pezzi della dama o degli scacchi si possono definire altre macro di posizionamento di diversi oggetti nelle caselle; ma mentre le definizioni presenti nel file di estensione `ScacchieraCB.sty` restano disponibili, esse possono servire da modello per creare altri comandi per collocare questi particolari oggetti; non se ne parla qui, perché esula dall'argomento di questo articolo.

La sintassi per creare questa scacchiera di base è fornita dal comando `\scacchierainterna` (vedi il codice 4 nella pagina 35)

1. Senza usare i comandi di ε-TeX, il comando per gestire i contatori TeX sarebbe `\advance \I by 1 \relax`; nel nostro caso non ci sarebbe una grossa differenza, ma per espressioni più complesse bisogna scrivere altri comandi che implicano caricare e scaricare i risultati intermedi in macro intermedie; il pacchetto calc offre un'interfaccia più comoda, ma il via vai fra le macro interne resta immutato. calc era utile e necessario prima che ε-TeX diventasse parte integrante degli interpreti del sistema TeX.

2. Quanto delimitato da graffe esplicite deve contenere un numero intero di graffe accoppiate `{}`, anche annidate, altrimenti l'interprete segnala un errore. Le situazioni insolite sono quando il gruppo viene aperto con una macro e chiuso con un'altra, situazioni che talvolta è necessario affrontare, ma non in questo pacchetto.

```
\scacchierainterna{⟨*⟩}[⟨largh⟩]%
  {⟨righe⟩,⟨colonne⟩}{⟨casella
scura⟩,⟨sfondo⟩}%
  (⟨x_off⟩,⟨y_off⟩)
```

i cui argomenti sono:

⟨*⟩ facoltativo; se presente genera la scacchiera di base, senza nulla nelle caselle; se assente genera la scacchiera di base con le caselle scure disposte come previsto dal bando di concorso.

⟨*largh*⟩ facoltativo; indica la larghezza esplicita dell'intera scacchiera; di default vale `0.7\linewidth`, per cui può essere usato sia per composizione a piena pagina, sia per la composizione in due colonne. Sulla base di questo valore e in base al numero di colonne della scacchiera viene calcolata la `\unitlength` dell'ambiente grafico picture.

⟨*righe*⟩ obbligatorio; specifica il numero N di righe dalla scacchiera.

⟨*colonne*⟩ facoltativo; se presente specifica il numero M delle colonne, se assente viene assunto pari a N; se è assente non è necessario usare la virgola di separazione.

⟨*casella scura*⟩ e ⟨*sfondo*⟩ facoltativi nonostante siano racchiusi fra graffe; se ⟨*casella scura*⟩ è presente specifica il colore delle caselle scure; di default è `black`; se non lo si vuole specificare, ma si vuole specificare il colore dello ⟨*sfondo*⟩, bisogna indicare anche la virgola; se non si vuole specificare lo ⟨*sfondo*⟩ si può specificare la virgola, ma non è necessario; il colore di default dello ⟨*sfondo*⟩ è `white`. Per i nomi dei colori si devono usare i nomi inglesi definiti dal pacchetto xcolor (Kern, 2016), o altri colori definiti mediante quel pacchetto; con xcolor, si possono usare le *color expressions*, ma negli argomenti del pacchetto scacchieraCB è necessario "nascondere" queste espressioni dentro un gruppo esplicito, perché i punti esclamativi che vi sono usati interferirebbero con il funzionamento di questo pacchetto; si scriverà pertanto `{blue!50!white}` invece di `blue!50!white`.

⟨x_{off}⟩ e ⟨y_{off}⟩ sono facoltativi, ma se specificati devono esserlo entrambi; servono per spostare l'intera scacchiera in alto con il valore ⟨y_{off}⟩ positivo, e a destra con ⟨x_{off}⟩ positivo. Di default essi valgono entrambi zero; essi vanno espressi in unità del grafico cioè come prefisso dell'unità di misura `\unitelength`

Come si capisce dal numero di informazioni facoltative, le possibilità di quelle presenti o assenti sono 7, quindi ci sono $2^7 = 128$ combinazioni. Ovviamente non sono tutte utili (per esempio chi specificherebbe `{blue,}` invece di `{blue}`? Chi specificherebbe `{,}` o `{}` invece di evitare completamente di esprimere l'opzione?), ma ci sono.

Si noti che `\scacchierainterna` è solo di servizio per i comandi e gli ambienti del prossimo paragrafo, infatti esso apre l'ambiente picture ma non lo chiude incondizionatamente; lo chiude solo se si usa il comando `\ComandoScacchiera`.

3.2 L'ambiente scacchiera

In apertura l'ambiente scacchiera imposta a `false` la variabile booleana `ScacCommand` e poi chiama il comando `\scacchierainterna` senza argomenti (argomenti che però `\scacchierainterna` legge come prima cosa in apertura dell'ambiente); in chiusura semplicemente chiude l'ambiente picture, quindi fra l'apertura e la chiusura si possono inserire ⟨*altri oggetti*⟩; la sintassi perciò è la seguente:

```
\begin{scacchiera}⟨argomenti
  per \scacchierainterna⟩
⟨altri oggetti⟩
\end{scacchiera}
```

Disegna una scacchiera con le caselle scure alternate con le caselle chiare (se non viene specificato l'asterisco, per il quale si veda il prossimo comando), come specificato dal bando; gli ⟨*altri oggetti*⟩ possono essere inseriti mediante i comandi `\GenBox` e, eventualmente, con `\ScacBox` e `\NumBox`, o con qualunque altro comando valido nell'ambiente picture.

3.3 L'ambiente cruciverba

L'ambiente cruciverba è molto simile all'ambiente scacchiera ma usa l'asterisco senza che debba farlo l'utente; non inserisce nessuna casella scura; la sintassi è la stessa:

```
\begin{cruciverba}⟨argomenti per
  \scacchierainterna senza asterisco⟩
⟨altri oggetti⟩
\end{cruciverba}
```

3.4 Le scatole speciali

Come già detto, sono disponibili tre scatole speciali con le seguenti sintassi:

```
\GenBox(⟨x,y⟩)[⟨posizione⟩](⟨dimensioni⟩)%
  {⟨oggetto⟩}
\NumBox(⟨x⟩,⟨y⟩)margnumero{⟨corpo⟩}
\Scacbox(⟨x⟩,⟨y⟩)
```

e con gli argomenti seguenti:

⟨*x,y*⟩ coppia di valori separati da virgola; obbligatori. Essi sono le coordinate dello spigolo inferiore sinistro della cella dove si vuole inserire qualche cosa. Di fatto sono numeri interi, perché l'unità di misura grafica è data proprio dalla lunghezza del lato del quadrato della cella. Si tenga presente che le coordinate della prima cella in basso a sinistra dell'intera scacchiera ha coordinate `(0,0)`.

⟨*posizione*⟩ facoltativa; è indicata da una coppia di lettere non inconsistenti scelte fra `l, r, t, b, c` per indicare dove si vuole mettere l'⟨*oggetto*⟩ dentro la cella. In sostanza sono gli stessi codici di posizione che si usano in ambiente picture come argomento facoltativo al comando `\makebox`. L'espressione "non inconsistenti" ricorda che ci sono degli accoppiamenti inconsistenti, come `[tb]`, perché un oggetto non può essere contemporaneamente allineato in alto e in basso.

⟨*dimensioni*⟩ facoltative; sono le dimensioni espresse in frazioni di `\unitlength` della scatola che contiene l'oggetto; di default valgono `(1,1)`. Per esempio si può specificare `(0.5,0.5)` per specificare una scatola (quadrata) il cui lato è la metà di quello della cella.

⟨*oggetto*⟩ obbligatorio; è quanto si vuole inserire in una cella; può essere qualunque cosa che l'ambiente picture accetti e sappia mettere in posizione.

⟨*numero*⟩ obbligatorio: è il ⟨*numero*⟩ che nei cruciverba va posto ad una piccola distanza dell'angolo superiore sinistro di una casella chiara.

⟨*corpo*⟩ facoltativo; è costituito dal coefficiente da dare a `\unitlength` per definire il corpo del ⟨*numero*⟩; di default vale `{0.3}`. È raro doversene servire, ma il comando che ne fa uso, `\setfontsize`, viene descritto nel prossimo paragrafo ed è a disposizione dell'utente, qualora lo volesse usare. Si veda la spiegazione di `\setfontsize` qui sotto.

3.5 Il corpo dei numeri dei cruciverba

Il corpo del numero delle celle numerate dei cruciverba, come si è visto sopra, si può scalare al valore desiderato.

Il comando che produce l'effetto ha la sintassi seguente:

`\setfontsize[`⟨*allargamento*⟩`]{`⟨*corpo*⟩`}`

La ⟨*dimensione del corpo*⟩ è una qualunque dimensione esplicita, per esempio, `{2mm}`, `{0.5\unitlength}`, e simili, o un numero reale senza l'indicazione dell'unità di misura che allora viene assunta di default pari al punto tipografico. L'⟨*allargamento*⟩ di default è impostato pari a `1.2`, ma opzionalmente l'utente può specificare valori un poco maggiori o minori. Il comando di fatto imposta l'⟨*allargamento*⟩ come fattore del parametro `\linespread` e imposta con `\fontsize` uno scartamento pari al corpo; poi dà il comando `\selectfont` che in pratica modifica lo scartamento per il fattore specificato con l'argomento facoltativo. I calcoli, quindi, vengono eseguiti dai comandi del nucleo di LaTeX. La definizione di `\setfontsize` è nelle ultime due righe del codice 4 nella pagina 35.

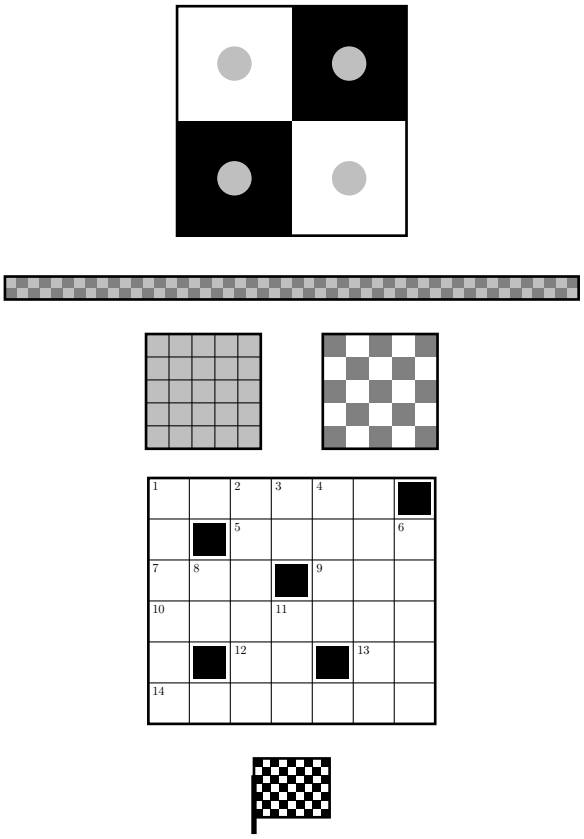


Figura 3: Alcuni esempi costruiti con il pacchetto scacchieraCB

Si noti, però, che questo comando opera correttamente solo con font continuamente scalabili, quali sono i font OpenType usati in questo testo, i Latin Modern, sempre consigliabili quando si compone con pdfLaTeX, i font Type 1 in generale. Non funziona invece con i font Computer Modern (semplici o super) a causa di come sono definiti i loro file di descrizione `.fd`. In realtà non sarebbe impossibile modificare questi file, ma cesserebbero di funzionare volendo ottenere il file finale in formato DVI con questi font; si veda in particolare Beccari (2017, figura 2.1).

3.6 Alcuni esempi

Nella figura 3 ci sono alcuni esempi costruiti con il software contenuto nel pacchetto scacchieraCB. Non sono esposti esempi colorati con colori vivaci, perché questa rivista viene stampata in bianco e nero e tonalità di grigio; d'altra parte l'uso dei colori 'gray' e 'lightgray' è sufficientemente esemplificativo, nonostante l'aspetto visivo di quei colori sia modesto.

1. Il primo e il quarto esempio mostrano due scacchiere costruite rispettivamente con l'ambiente scacchiera e il comando `\ComandoScacchiera` senza usare l'asterisco; nel primo si è usato il comando `\GenBox` per sovrapporre in tutte le celle il pallino grigio; i codici sono i rispettivamente i seguenti:

```
\ProvidesPackage{ScacchieraCB}[2018-02-10 v.1.0 Per creare scacchiere e cruciverba]
%
\providecommand*\usecs[1]{\csname#1\endcsname}
\providecommand*\whilenum[2]{\usecs{@whilenum}#1\do{#2}}
%
\newif\ifScacCommand \ScacCommandfalse
\newcommand\ComandoScacchiera{\bgroup\ScacCommandtrue\scacchierainterna}
\newenvironment{cruciverba}{\ScacCommandfalse\scacchierainterna*}{\end{picture}}
\newenvironment{scacchieraCB}{\ScacCommandfalse\scacchierainterna}{\end{picture}}
%                                                              Scacchiera di base
\NewDocumentCommand\scacchierainterna{s O{0.7\linewidth} m G{black,white} D(){0,0}}{%
%                                                                   Registri
\countdef\N=256   \countdef\M=260%
\countdef\I=258   \countdef\J=262%
\countdef\No=264  \countdef\Mo=266
\def\splitOffsets##1,##2!{\No=##1\relax\Mo=##2}
\splitOffsets#5!%
\def\toglivirgola##1,!{##1}%
\def\righecolonne##1,##2!{\M=##1\relax%                          \M righe e \N colonne
\def\ScacA{##2}\relax
\ifx\ScacA\empty
   \N=\M\relax
\else
   \expandafter\N\toglivirgola##2!\relax
\fi
\ifnum\M<1\relax\M=1\fi
\ifnum\N<1\relax\N=1\fi}%
\def\colori##1,##2!{%
  \def\scacco{##1}\ifx\scacco\empty
  \def\scacco{\color{black}}\else
  \def\scacco{\color{##1}}\fi
  \def\sfondo{##2}\ifx\sfondo\empty\def\sfondo{\color{white}}\else
  \ifdefstring{\sfondo}{,}{\def\sfondo{\color{white}}}%
  {\edef\sfondo{\noexpand\color{\toglivirgola##2!}}}\fi
}%
\def\Colori{#4}%
\ifx\Colori\empty\colori,!\else\colori#4,!\fi
%
\righecolonne#3,!%
\dimendef\Num=256\dimendef\Den=258%
\Num=1pt\Den=\N\Num
\unitlength=\dimexpr #2*\Num/\Den\relax
\I=\numexpr\N+\No\relax
\J=\numexpr\M+\Mo\relax
%                    Disegno della griglia con contorno e sfondo con o senza contenuti
\noindent\begin{picture}(\I,\J)(-\No,-\Mo)
\put(0,0){\sfondo\polygon*(0,0)(\N,0)(\N,\M)(0,\M)}%
\IfBooleanTF{#1}{%                                    Se ''true'' si fa cruciverba
\def\nero{\scacco\polygon*(0.1,0.1)(0.9,0.1)(0.9,0.9)(0.1,0.9)}%
\multiput(0,0)(1,0){\numexpr\N+1}{\line(0,1){\M}}
\multiput(0,0)(0,1){\numexpr\M+1}{\line(1,0){\N}}
}{%                                                altrimenti si fa scacchiera
\def\nero{\scacco\polygon*(0,0)(1,0)(1,1)(0,1)}%
\I=0\J=0\relax
\whilenum{\N > \I}{{\whilenum{\M > \J}{\unless
\ifodd\numexpr\I+\J \put(\I,\J){\nero}\fi
\J=\numexpr\J+1\relax}}\I=\numexpr\I+1\relax}}%
\put(0,0){\linethickness{1pt}\polygon(0,0)(\N,0)(\N,\M)(0,\M)}
\ifScacCommand\end{picture}\egroup\fi}
%                                                                      Scatole
\NewDocumentCommand\GenBox{ D(){0,0} O{cc} D(){1,1} m }{%
  \put(#1){\makebox(1,1)[bl]{\makebox(#3)[#2]{#4}}}}
\NewDocumentCommand\NumBox{D(){0,0} m G{0.3}}{%
  \put(#1){\makebox(1,1)[cc]{\makebox(0.8,0.8)[tl]{\setfontsize{#3\unitlength}#2}}}}
\def\ScacBox(#1){\put(#1){\makebox(1,1)[bl]{\nero}}}
%                                                                 Corpo dei font
\newcommand*\setfontsize[2][1.2]{%
\linespread{#1}\fontsize{#2}{#2}\selectfont}
```

Codice 4: Codice della soluzione di Beccari

```
\begin{scacchiera}[0.4\linewidth]{2}
  \GenBox(0,0){\color{lightgray}\circle*{0.3}}
  \GenBox(1,1){\color{lightgray}\circle*{0.3}}
  \GenBox(1,0){\color{lightgray}\circle*{0.3}}
  \GenBox(0,1){\color{lightgray}\circle*{0.3}}
\end{scacchiera}

\ComandoScacchiera[0.2\linewidth]{5}%
    {,lightgray}
```

2. Il secondo ed il terzo esempio sono creati entrambi con `\ComandoScacchiera` ma nel terzo si è usato l'asterisco, mentre nel secondo si sono specificate 2 righe e 50 colonne. I codici sono rispettivamente i seguenti:

```
\ComandoScacchiera[\linewidth]{2,50}%
    {gray,lightgray}

\ComandoScacchiera*[0.2\linewidth]{5}%
    {,lightgray}
```

3. Il quinto esempio mostra un cruciverba; il suo codice è piuttosto lungo, perché ogni casella nera e ogni numerino va introdotto con un comando apposta, in quanto le coordinate delle caselle dove mettere quegli oggetti non sono determinate a priori.

```
\begin{cruciverba}[0.5\linewidth]{6,7}
  \ScacBox(1,1)   \ScacBox(3,3)
  \ScacBox(1,4)   \ScacBox(4,1)
  \ScacBox(6,5)
  \NumBox(0,5){1} \NumBox(1,3){8}
  \NumBox(2,5){2} \NumBox(4,3){9}
  \NumBox(3,5){3} \NumBox(0,2){10}
  \NumBox(4,5){4} \NumBox(3,2){11}
  \NumBox(2,4){5} \NumBox(2,1){12}
  \NumBox(6,4){6} \NumBox(5,1){13}
  \NumBox(0,3){7} \NumBox(0,0){14}
\end{cruciverba}
```

4. Il sesto esempio mostra quella bandiera con il suo manico che viene sventolata nelle corse di Formula 1 al passaggio per il traguardo finale. Potrebbe essere un'alternativa (insolita) all'uso del Q.E.D. alla fine delle dimostrazioni matematiche. Il codice è il seguente:

```
\begin{scacchiera}[10mm]{7,9}
\put(0,-2,){\makebox(0,0)[bl]{%
   \linethickness{2pt}\line(0,1){7}}}
\end{scacchiera}
```

   e vi si mostra l'uso di comandi propri dell'ambiente picture all'interno dell'ambiente.

4 Intermezzo

Vale la pena di ricordare che i programmi `latex`, `pdflatex`, `xelatex` e `lualatex` non disegnano direttamente nulla; preparano un file di uscita che contiene dei comandi nativi di TEX, `\special`, il cui argomento contiene, appunto, informazioni 'speciali' destinate ad istruire i programmi di stampa o di visualizzazione di eseguire i disegni. Sia TikZ sia pict2e, usati da Molinaro e da Beccari, non fanno altro che trasformare le macro del linguaggio TEX o LATEX in istruzioni per quei programmi di visualizzazione e di stampa. Di per sé il formato DVI originale di Knuth o quello esteso usato da XeLATEX affidano la loro visualizzazione ad altri programmi; per semplicità diciamo che il programma `dvipdfmx` traduce quei dati speciali in istruzioni del linguaggio PDF nel momento in cui il formato DVI viene trasformato in quello PDF, mentre i programmi `pdflatex` e `lualatex` producono direttamente il formato PDF.

I pacchetti TikZ e `pict2e`, quindi, non sono altro che delle interfacce fra l'utente e il linguaggio PDF per trasformare le istruzioni di composizione, comprese quelle per il disegno, in istruzioni inserite nel file finale ma destinate al programma di visualizzazione.

5 La soluzione di Giacomelli

La soluzione di Giacomelli, che viene descritta in questo paragrafo, sfrutta il linguaggio Lua per inserire direttamente le istruzioni di disegno nel file PDF prodotto da `lualatex` senza ricorrere a nessun pacchetto di interfaccia. In realtà la soluzione di Giacomelli prima crea poi usa una libreria in linguaggio Lua che estende quanto è già contenuto dentro il compilatore `luatex`.

Il lettore che voglia approfondire il linguaggio Lua ha a disposizione il testo originale del suo creatore (Ierusalimschy, 2016) e il manuale di istruzioni del compilatore `luatex`, (The LuaTEX development team, 2017).

5.1 Lo scopo del codice

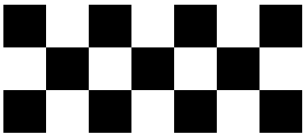
Lo scopo del codice di Giacomelli è quello di creare un comando `\scacchieraRG` che accetti due argomenti con la sintassi seguente:

`\scacchieraRG{`⟨*colonne*⟩`}{`⟨*righe*⟩`}`

che possa essere usato per disegnare una scacchiera conforme ai requisiti del concorso, salvo il fatto che è in grado di disegnare una scacchiera rettangolare con numeri di ⟨*righe*⟩ e ⟨*colonne*⟩ specificate "a piacere"; per esempio il codice[3]:

```
\scacchieraRG{7}{3}
```

e produce la figura:



Quello che Giacomelli si propone di fare è scrivere direttamente nel file di uscita le istruzioni in linguaggio PDF attraverso il linguaggio Lua.

3. Si ricorda che in questo articolo i nomi delle macro sono stati estesi con le iniziali dei rispettivi autori proprio per non generare conflitti.

5.2 I comandi primitivi del linguaggio PDF

Per conoscere questi comandi primitivi del linguaggio PDF, il protocollo pubblico della Adobe è veramente troppo lungo e complesso, ma il documento Hàn Thế Thành *et al.* (2017) contiene il necessario. Qui si ricorda solo quali siano i comandi nativi di PDF per disegnare un rettangolo:

⟨*x*⟩ ⟨*y*⟩ ⟨*w*⟩ ⟨*h*⟩ `re`

dove ⟨*x*⟩ e ⟨*y*⟩ rappresentano le coordinate del vertice inferiore sinistro; ⟨*w*⟩ e ⟨*h*⟩ la larghezza e l'altezza.

Per non confondere le istruzioni di disegno è bene che ogni oggetto sia racchiuso dentro un stack insieme allo spessore delle linee del disegno e al modo di congiungersi delle spezzate che compongono l'eventuale contorno. Quindi al codice di base indicato sopra bisogna aggiungere qualche altro comando; in particolare tutto va incluso nell'argomento del comando `\pdfliteral`, tanto che il codice seguente

```
\pdfliteral{
q
0   0 32 32 re
32 32 32 32 re
f
S
Q
}
```

inserito all'interno di un documento `.tex`, permette di disegnare due rettangoli (quadrati) neri, uno con il vertice nella coordinata `(0,0)` e l'atro con il vertice nella coordinata `32,32`, entrambi aventi base e altezza uguali a 32 unità. I codici `q` e `Q` servono per aprire e chiudere uno stack (rispettivamente gli equivalenti di `push` e `pop`) e gli altri per le altre caratteristiche del disegno: `f` per `fill` e `S` per `stroke`. Come si vede il codice è compattissimo, anche se sostanzialmente si tratta delle stesse cose, scritte in modo meno prolisso, che si possono scrivere in PostScript.

Queste unità, sia ben chiaro, devono essere i "big point", i punti PostScript, con la sigla TEX `bp`, raramente usati direttamente dagli utenti, ma richiesti espressamente dal linguaggio PDF[4].

Quello che si ottiene con quel codice è semplicemente una piccola scacchiera di 2 righe e due colonne:

4. Anche TEX di default usa i punti tipografici e non i punti PostScript; la differenza è minima ma c'è: infatti 1 pt = (1/72,27) in ≈ 0,35146 mm mentre 1 bp = (1/72) in ≈ 0,35278 mm con una differenza di un poco meno del 0,4%, piccola ma non trascurabile. Ne consegue che quei 32 bp sono circa 10 mm.

Tuttavia il codice mostrato sopra non à ancora completo, perché non occupa lo spazio corrispondente alle sue dimensioni effettive. Vedremo fra poco come fare.

5.3 La funzione Lua `build_pdfliteral()`

Ora entra in scena il linguaggio Lua; la funzione Lua `build_pdfliteral()` ha il compito di generare il codice che descrive tutti i quadrati della scacchiera. Perciò dovrà ricevere in ingresso il numero di caselle in righe e colonne.

La soluzione ideale per produrre la stringa di descrizione `pdfliteral` è usare una tabella Lua come buffer e poi concatenare gli elementi con la funzione di libreria `table.concat()`. Dovremo inoltre rispettare il vincolo "casella in basso a sinistra sempre nera".

Per considerare l'alternanza di caselle nere e bianche è possibile far uso di una variabile booleana nominata per esempio `is_black`:

```
-- w : numero di caselle in orizzontale
-- h : numero di caselle in verticale
-- hs: misura in big point della singola casella
local function build_pdfliteral(w, h, hs)
  local y = 0
  local is_odd = true
  local fmt = string.format
  local fdim = fmt(''%d %d re'', hs, hs)
  local fret = ''%d %d '' .. fdim

  local t = {}
  t[#t + 1] = ''q''

  for _ = 1, h do
    local is_black = is_odd
    local x = 0
    for _ = 1, w do
      if is_black then
          t[#t + 1] = fmt(fret, x, y)
          is_black = false
      else
          is_black = true
      end
      x = x + hs
    end
    y = y + hs
    is_odd = not is_odd
  end

  t[#t + 1] = ''f''
  t[#t + 1] = ''S''
  t[#t + 1] = ''Q''
  return table.concat(t, ''\n'')
end
```

Il cuore della funzione è basato su due cicli `for` annidati. Per i più curiosi, si è utilizzata un'espressione idiomatica di Lua per riempire in sequenza l'array `t`, usando l'operatore `#` che restituisce la dimensione dell'oggetto. Così l'intero `#t + 1` è l'indice della prossima posizione libera nell'array.

Altra nota interessante da evidenziare sul linguaggio Lua riguarda l'assegnazione della funzione `string.format()` alla variabile `fmt` in una delle

prime righe del codice. Non si tratta solamente della comodità dell'abbreviazione del nome, ma dell'efficienza di esecuzione del codice perché l'accesso alle variabili locali è più veloce di quello alle variabili globali. La cosa può sorprendere chi non conosca questo linguaggio: in Lua le funzioni sono oggetti di prima classe, cioè sono valori assegnabili alle variabili come qualsiasi altro tipo.

Eseguendo la funzione in uno script Lua per riprodurre il codice iniziale, potremo verificare che effettivamente essa produce il risultato corretto. Per esempio la chiamata:

```
print(build_pdfliteral(2, 2, 32))
```

fornisce il codice pdfliteral dell'esempio precedente (che qui si ripete):

```
q
0 0 32 32 re
32 32 32 32 re
f
S
Q
```

Tuttavia, invece di verificare ogni volta il flag `is_black`, si potrebbe rendere un po' più efficiente la funzione considerando che su ciascuna riga della scacchiera è sufficiente raddoppiare l'incremento della coordinata $\langle x \rangle$ e disegnare sempre la casella come nera:

```
local function build_pdfliteral(w, h, hs)
    local y = 0
    local is_odd = true
    local fmt = string.format
    local fdim = fmt(''%d %d re'', hs, hs)
    local fret = ''%d %d '' .. fdim

    local t = {}
    t[#t + 1] = ''q''

    for _ = 1, h do
        local start = is_odd and 1 or 2
        local x = is_odd and 0 or hs
        for _ = start, w, 2 do
            t[#t + 1] = fmt(fret, x, y)
            x = x + 2*hs
        end
        y = y + hs
        is_odd = not is_odd
    end

    t[#t + 1] = ''f''
    t[#t + 1] = ''S''
    t[#t + 1] = ''Q''
    return table.concat(t, ''\n'')
end
```

Ma c'è una terza soluzione: invece di pensare alla scacchiera come a un insieme semplice di quadrati, essa può essere considerata come la sovrapposizione di due griglie regolari traslate una rispetto all'altra del vettore corrispondente alla diagonale

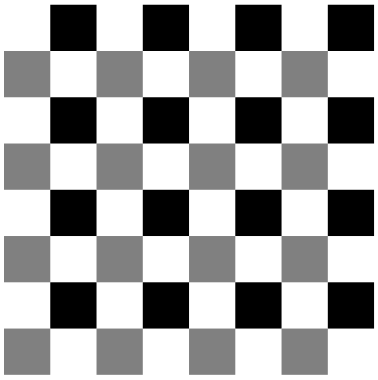


FIGURA 5: Schema della scacchiera a otto caselle che dimostra come la si possa disegnare sovrapponendo due diverse griglie regolari di quadrati, quella grigia e quella nera. Questa soluzione è quella utilizzata per implementare il comando `\scacchieraRG`.

di una casella. In questo modo il codice diviene più semplice e comprensibile.

Si definisca allora la funzione ausiliaria `grid()` che scriva in un buffer il codice grafico dei quadrati di una griglia regolare di passo doppio della dimensione delle caselle, a partire dalla posizione 1 oppure 2. Se si indica per `start_pos` il valore 1 si intende disegnare la griglia la cui casella in basso a sinistra è nella posizione `(1, 1)` , mentre con il valore 2 si intende la seconda griglia sovrapposta che inizia dalla casella nella posizione `(2, 2)`:

```
local function grid(buf, hs, w, h, start_pos)
    assert(start_pos == 1 or start_pos == 2)
    local fmt = string.format
    local fdim = fmt(''%d %d re'', hs, hs)
    local fret = ''%d %d '' .. fdim

    local delta = 2 * hs
    local y = start_pos == 1 and 0 or hs
    local x_0 = y
    for _ = start_pos, h, 2 do
        local x = x_0
        for _ = start_pos, w, 2 do
            buf[#buf + 1] = fmt(fret, x, y)
            x = x + delta
        end
        y = y + delta
    end
end
```

Nel listato, per calcolare il valore iniziale dell'ordinata $\langle y \rangle$ si è fatto ricorso all'equivalente Lua dell'operatore ternario del C++. Con questa funzione, quella principale si riduce alla seguente:

```
local function build_pdfliteral(w, h, hs)
    local t = {}
    t[#t + 1] = ''q''
    grid(t, hs, w, h, 1)
    grid(t, hs, w, h, 2)
    t[#t + 1] = ''f''
    t[#t + 1] = ''S''
    t[#t + 1] = ''Q''
    return table.concat(t, ''\n'')
end
```

Per la scacchiera 8×8 della figura 5, i quadrati grigi sono disegnati dalla prima chiamata a `grid()`, mentre con la seconda sono disegnati quelli colorati in nero.

5.4 La funzione `create_box()`

Con la seconda funzione si passa da Lua a LuaTEX. Useremo le librerie interne `node` per creare il nodo `pdfliteral`, e `tex` per inserire il nodo in un registro di tipo box. Non occorrono particolari aggiustamenti di *bounding box* perché i limiti inferiore e sinistro del disegno coincidono con gli assi coordinati:

```
local function create_box(boxname, w, h)
  -- fixed house dimension
  local house_size = 16
  -- pdfliteral node
  local n = node.new(
    ''whatsit'', ''pdf_literal''
  )
  n.data=build_pdfliteral(w, h, house_size)
  assert(
    tex.isbox(boxname),
    string.format(
     ''Box register [%s] doesn't exist'',
     boxname
    )
  )
  -- horizontal box
  local hbox = node.hpack(n)
  local dim_sp = tex.sp(
      tostring(house_size)..''bp''
  )
  hbox.width  = w * dim_sp
  hbox.height = h * dim_sp
  tex.box[boxname] = hbox
end
```

Le funzioni Lua possono essere inserite in un file esterno con estensione `.lua`. In questo modo non dovremo gestire la sovrapposizione dei diversi significati assegnati da TEX e da Lua ad alcuni caratteri speciali come il simbolo del percento. In Lua esso è l'operatore modulo, oppure la definizione delle classi di formato; in TEX è il simbolo di commento.

Il modo più semplice per creare un modulo di libreria in Lua è creare una tabella contenitore delle funzioni, restituendone il riferimento con un'istruzione finale `return`. In realtà solamente la funzione `create_box()` può essere considerata come pubblica, mentre la `build_pdfliteral()` e la sua ausiliaria `grid()` possono essere considerate come funzioni private non accessibili dall'utente.

Per fare questo definiremo come locali le ultime due, contando sul meccanismo della *closure*, perché continuino a essere accessibili internamente, mentre per la prima useremo la speciale sintassi seguente per memorizzare direttamente in un campo della tabella la funzione pubblica:

function ⟨*table_name*⟩.⟨*func_name*⟩(⟨*argomenti*⟩)
  – ⟨*body*⟩
end

Il codice completo è riportato nella pagina 40.

5.5 Il risultato finale

Ora tutto è disponibile per disegnare la scacchiera. Si scrive il sorgente LuaLATEX che carica le funzioni dal file esterno `libsk.lua` posizionato nella stessa directory[5] e definisce la nuova macro `\scacchiera`. Quest'ultima passa subito il controllo a Lua che riceve in ingresso i valori delle dimensioni della scacchiera da disegnare assegnandoli a due variabili locali, per effetto dell'espansione.

Dopo viene chiamata la funzione `create_box()`. Al termine di questo passaggio nel registro `\libsk` ci sarà il disegno come se lo avessimo creato con la primitiva `\pdfliteral`. Perciò non ci rimane che utilizzare il registro tramite l'istruzione TEX `\box`.

Nel listato qui di seguito si è riportato il codice, ridondante per gli scopi principali, ma utile per mostrare al lettore come appare il preambolo di un tipico documento LuaLATEX. Rispetto a pdfLATEX non si carica più il pacchetto inputenc perché il sorgente deve essere codificato UTF-8, mentre si usa fontspec per l'impostazione dei font – qui, a titolo di esempio, si utilizzano il font Open Type Libertinus – e si sostituisce babel con polyglossia.

Ultima osservazione, si è inserito prima e dopo il disegno una lettera 'A' per verificare che la costruzione della scatola sia corretta e corrisponda alle dimensioni del disegno. Il risultato è mostrato nella figura 7.

```
% !TeX program = LuaLaTeX
\documentclass{article}

% font setup
\usepackage{fontspec}
\defaultfontfeatures{Ligatures=TeX}
\setmainfont{Libertinus Serif}
\setmonofont{Libertinus Mono}

% language setup
\usepackage{polyglossia}
\setmainlanguage[babelshorthands]{italian}

% si carica la libreria libsk.lua
\directlua{
    libsk = require ''libsk''
}
% si definisce il comando \scacchiera
\newbox\libskbox
\newcommand\scacchiera[2]{%
\directlua{
```

5. Usare `libsk.lua` per comporre documenti differenti con i relativi file conservati in cartelle differenti rende necessario copiarlo in altrettante cartelle. Si può evitare questa cosa in due modi: salvandolo in una cartella del proprio albero personale oppure inserendo nelle varie cartelle solo un link simbolico all'unica copia del file anzidetto. I link simbolici sono da sempre disponibili nei sistemi operativi Linux e Mac, e lo sono anche con i sistemi Windows da Win 8 in poi.

```
local libsk = {}

local function grid(buffer, house_size, w, h, start_pos)
    assert(start_pos == 1 or start_pos == 2)
    local fmt = string.format
    local fdim = fmt(''%d %d re'', house_size, house_size)
    local fret = ''%d %d '' .. fdim
    local y = start_pos == 1 and 0 or house_size
    local x_0 = y
    local delta = 2 * house_size
    for _ = start_pos, h, 2 do
        local x = x_0
        for _ = start_pos, w, 2 do
            buffer[#buffer + 1] = fmt(fret, x, y)
            x = x + delta
        end
        y = y + delta
    end
end

local function build_pdfliteral(w, h, house_size)
    local t = {}
    t[#t + 1] = ''q''
    grid(t, house_size, w, h, 1)
    grid(t, house_size, w, h, 2)
    t[#t + 1] = ''f''
    t[#t + 1] = ''S''
    t[#t + 1] = ''Q''
    return table.concat(t, ''\n'')
end

function libsk.create_box(boxname, w, h)
        -- fixed house dimension
        local house_size = 16
    -- pdfliteral node
    local n = node.new(
        ''whatsit'', ''pdf_literal''
    )
    n.data = build_pdfliteral(w, h, house_size)

    assert(
        tex.isbox(boxname),
        string.format(
           ''Box register [%s] doesn't exist'',
           boxname
        )
    )

    -- horizontal box
    local hbox = node.hpack(n)
    local dim_sp = tex.sp(tostring(house_size)..''bp'')
    hbox.width  = w * dim_sp
    hbox.height = h * dim_sp
    tex.box[boxname] = hbox
end

return libsk
```

Codice Lua 6: Codice da salvare in un file luask.lua che deve essere conservato in una cartella come viene spiegato nel testo.

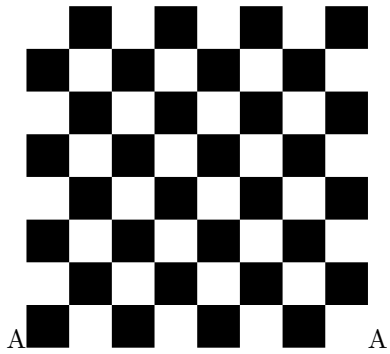


FIGURA 7: Il risultato finale della scacchiera creata con LuaTEX. Le lettere 'A' prima e dopo il disegno mostrano il corretto dimensionamento della scatola che contiene il disegno.

```
    local w = tonumber(#1)
    local h = tonumber(#2)
    libsk.create_box([[libskbox]], w, h)
}%
\leavevmode
\box\libskbox
}

\begin{document}
A\scacchieraRG{8}{8}A
\end{document}
```

5.6 Facile o difficile?

Osservando il codice Lua del file `libsk.lua` si potrebbe pensare che sia *difficile* scriverlo e quindi verrebbe voglia di rinunciarvi. Secondo l'opinione di Giacomelli, Lua è in realtà molto più semplice di TEX e la soluzione descritta non fa uso di nessun pacchetto esterno, ma tutto il lavoro è fatto internamente a LuaLATEX.

6 Confronti e commenti finali

Alla fine di questa carrellata sulle tre soluzioni mostrate, molto diverse l'una dall'altra, si possono fare alcuni confronti.

Non c'è dubbio che la più semplice sia quella di Molinaro, la più versatile sia quella di Beccari e la più professionale quella di Giacomelli.

I codici sono molto diversi: quelli di Molinaro e di Giacomelli attuano con i rispettivi strumenti un metodo originale per comporre la scacchiera sovrapponendo due scacchiere sfalsate in diagonale di una casella contenenti caselle di colore scuro con passo 2, ma i metodi seguiti sono diversi; entrambi in un certo senso dimensionano queste due scacchiere in modo che siano basate su dimensioni pari a quelle specificate, modulo 2. Siccome gli strumenti disponibili sono diversi, il modo di attuare in pratica questa operazione modulo 2 è molto diversa.

La soluzione di Beccari usa invece una sola scacchiera, ma usa un algoritmo originale per decidere se in una data casella si debba inserire una casella scura; tuttavia, siccome Beccari pensa anche alla creazione di cruciverba, sembra abbastanza ovvio che non si sia preoccupato di escogitare un "trucco" così efficace come quello delle due scacchiere sovrapposte e sfalsate.

La versatilità si vede anche confrontando gli argomenti dei comandi o degli ambienti che creano le rispettive scacchiere. Abbiamo le seguenti differenze:

Molinaro

```
{⟨righe⟩}{⟨dimensione casella⟩}
```

Beccari

1. La specificazione più semplice:

   ```
   {⟨righe⟩}
   ```

2. La specificazione più complessa:

   ```
   ⟨*⟩[⟨larghezza scacchiera⟩]%
     {⟨righe⟩,⟨colonne⟩}{⟨colori⟩}%
       [⟨offset⟩]
   ```

Giacomelli

```
{⟨colonne⟩}{⟨righe⟩}
```

È altrettanto ovvio che se le soluzioni di Molinaro e di Giacomelli avessero la stessa versatilità di quella di Beccari, i loro codici dovrebbero allungarsi moltissimo, specialmente se gli autori volessero rimanere completamente dentro la loro impostazione di base, TikZ per Molinaro e `libsk.lua` per Giacomelli; ma i confronti non si devono fare su questo livello, dato che il concorso non lo prevedeva. Non prevedeva nemmeno che l'intera scacchiera avesse un contorno evidente, ma Beccari e Molinaro hanno provveduto in merito, mentre Gacomelli non l'ha previsto.

Invece il lettore ha modo di imparare diversi approcci e diversi modi di affrontare lo stesso problema; l'analisi critica di questi approcci e dei diversi modi esposti permette di usare la strategia più opportuna quando il lettore dovesse affrontare problemi la cui soluzione non sia così ovvia come potrebbe sembrare a prima vista.

7 Epilogo

I tre autori hanno livelli di dimestichezza con LATEX molto diversa; Maurizio Molinaro è socio del GUIT dal 2014; il suo contributo più importante e visibile alla comunità degli utenti è il bell'articolo (MOLINARO, 2017) sulla tipografia relativa al gioco degli scacchi. In un certo senso ci si sarebbe stupiti se l'autore non avesse presentato una sua soluzione al problema del concorso.

Claudio Beccari lavora con LATEX da più di 30 anni; ha una grandissima esperienza alle spalle, ma è molto affezionato alle cose tradizionali, una delle quali è l'ambiente picture sul quale ha

investito molte energie, non solo collaborando allo sviluppo di pict2e ma anche costruendo per suo uso e consumo grossi pacchetti basati su pict2e per estenderne le funzionalità al disegno di schemi particolari, come i circuiti elettronici e i disegni di diagrammi e istogrammi.

Roberto Giacomelli lavora con LaTeX da una dozzina di anni e lo usa abitualmente per la sua attività professionale; recentemente ha pubblicato diversi articoli su ArsTEXnica dove mostra le funzionalità di cui dispone LuaLaTeX per venire incontro a moltissimi problemi che si presentano nella documentazione relativa all'esercizio della sua professione.

Riferimenti bibliografici

Beccari, C. (2017). «Font e tipografia – pdfLaTeX, XeLaTeX, LuaLaTeXe i font». PDF document. Scaricabile da `http://www.guitex.org/home/images/doc/GuideGuIT/guidafont.pdf`.

Fischer, U. (2014). *chessboard: A package to print chessboards*. Versione 1.7; documento PDF leggibile con `texdoc chessboard`.

Gässlein, H., Niepraschk, R. e Tkadlec, J. (2016). «The pict2e package». PDF document. Leggibile con `texdoc pict2e`.

Hàn Thế Thành, Rahtz, S., Hagen, H., Henkel, H., Jackowski, P. e Schröder, M. (2017). «The pdfTeX user manual». PDF document. Leggibile con `texdoc pdftex-a`.

Ierusalimschy, R. (2016). *Programming in Lua*. Lua.org, 3ª edizione.

Kern, U. (2016). «Extending LaTeX's color facilities: the xcolor package». PDF document. Leggibile con `texdoc xcolor`.

Lamport, L. (1994). *LaTeX. A Document Preparation System*. Addison-Wesley, 2ª edizione.

Lehman, P. e Wright, J. (2017). «The etoolbox package». PDF document. Leggibile con `texdoc etoolbox`.

Molinaro, M. (2017). «Introduzione alla composizione di testi scacchistici». *ArsTEXnica*, (25).

The LaTeX3 project (2017). «The xparse package». PDF document. Leggibile con `texdoc xparse`.

The LuaTeX development team (2017). *LuaTeX Reference Manual*, 1.0.4 edizione.

The NTS Team (1998). «The ε-TeX manual». PDF document. Leggibile con `texdoc etex`.

Thorup, K. K., Jensen, F. e Rowley, C. (2014). «The calc package». PDF document. Leggibile con `texdoc calc`.

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

▷ Roberto Giacomelli
`giaconet dot mailbox at gmail dot com`

▷ Maurizio Molinaro
`maurizio dot molinaro3 at gmail dot com`

Un mediafilter LaTeX per DSpace

Emmanuele Somma

Sommario

L'articolo presenta un semplice programma di filtraggio (*mediafilter*) per permettere la trasformazione in PDF di pubblicazioni LaTeX depositate in un repository DSpace. La soluzione presentata permette l'integrazione nella pubblicazione anche dei metadati registrati nella scheda archivistica, che oltretutto possono essere utilizzati come caratteristiche dell'impaginazione o per guidare la compilazione.

Abstract

This article shows a simple filtering program (*mediafilter*) to turn LaTeX sources stored in a DSpace repository into PDF. The solution also allows to use metadata, stored in the archive cards, into the publication itself, as parameters for the publishing layout or the compilation process.

1 Introduzione

DSpace è una piattaforma di archiviazione e gestione degli oggetti digitali (articoli, libri, tesi, dataset, ecc.) usata come repository istituzionale o come servizio archivistico da enti e organizzazioni, in particolare in campo universitario o governativo (De Robbio (2002)).

DSpace è un programma server-side Java che interagisce con gli utenti attraverso un'interfaccia web e offre agli utenti le tipiche operazioni di navigazione nel contenuto e ricerca delle schede archivistiche. È programmabile, molto configurabile ed estendibile aggiungendo moduli scritti in linguaggio Java o Jython. Chi vuole contribuire alla collezioni documentali può registrare in autonomia nuove schede e gli amministratori possono gestire flussi di lavorazione, anche complessi, attraverso l'interfaccia amministrativa del repository, delle collezioni e delle comunità (collezioni di collezioni). Esiste anche un insieme di programmi accessibili sulla linea di comando del server che permette agli amministratori di compiere le operazioni più sofisticate come importazioni ed esportazioni, reindicizzazioni, migrazioni e molte altre.

Le schede archivistiche registrate in DSpace si riferiscono agli *elementi* (in inglese *item*) e collezionano un esteso insieme di metadati in formato Dublin Core (e altri schemi aggiuntivi). Ad ogni elemento sono collegati i file da conservare (denominati *bitstream* per sottolineare l'indipendenza dal formato).

DSpace non prevede che i file siano conservati con uno specifico formato (ad esempio PDF o DOC), dunque la scelta del tipo dei file dipende dalla politica degli amministratori del repository in merito all'uniformità del contenuto, economia gestionale e fruibilità dell'archivio. Si pone quindi il problema di fare delle scelte, di natura essenzialmente editoriale, su quali file conservare nell'insieme di quelli utili.

Solitamente sono conservati i file definitivi pubblicati (è il caso delle tesi, ad esempio), oppure i preprint (nel caso delle pubblicazioni su riviste esterne) o addirittura scansioni (nel caso di libri storici). Negli ultimi tempi i repository sono usati anche per conservare i dataset usati nella ricerca e persino il software realizzato.

L'articolo presenta il caso ipotetico di una collezione documentale per cui i gestori hanno deciso di offrire agli utenti finali l'accesso alle pubblicazioni in un formato comodo per la fruizione (ad esempio PDF), ma di conservare come fonte primaria del documento il sorgente LaTeX. È compito del server DSpace trasformare questo sorgente in un documento PDF.

Questa trasformazione *automatica* è gestita attraverso la realizzazione di un breve programma Java messo all'interno dell'insieme dei cosiddetti *mediafilter* di DSpace (Donohue (2016)). Viene prima proposta una soluzione minimale e successivamente considerate alcune semplici estensioni.

2 I *mediafilter* di DSpace

Un *mediafilter* è un *oggetto* dell'applicazione DSpace, derivato dalla classe MediaFilter, che agisce su un dato formato di *bitstream* in ingresso (che nel caso in esame è un file LaTeX con estensione .tex) e produce un bitstream in uscita (qui un file .pdf prodotto dalla compilazione con pdflatex).

All'interno del codice sorgente di un *mediafilter* devono essere definiti i metodi dell'oggetto derivato dalla classe MediaFilter che realizzano la trasformazione del bitstream dall'ingresso all'uscita; in questo senso si può dire che il programma sia un *filtro* da un *media* all'altro (*mediafilter* appunto). Se l'operazione è eseguita senza errori, il bitstream prodotto è registrato nella scheda archivistica insieme a quello d'ingresso con un nome, una descrizione e un tipo adeguato.

È possibile mettere in esecuzione un *mediafilter* manualmente con il comando :

```
1 [dspace]/bin/dspace filter -media \
```

```
                                -i <element>
```

oppure attraverso un processo temporizzato (cron).

Se tra i bitstream presenti in un *elemento* non si trova quello nel formato prodotto dal *mediafilter* in esame (o se è usata sulla linea di comando l'opzione `-f` o `-force`) allora il *mediafilter* viene messo in esecuzione e viene tentata la generazione automatica.

La creazione di un *mediafilter* avviene attraverso l'estensione della classe `MediaFilter` presente nel package `org.dspace.app.mediafilter` con la definizione obbligatoria di alcuni metodi:

```
1  public class Tex2PdfFilter extends
       Mediafilter {
2    public String getFilteredName(String
         oFileName) {
3      // Definisce il nome del file di
           output
4      return  oldFilename.replaceFirst(".
           tex$",".pdf");
5    }
6    public String getBundleName() {
7      // Definisce il nome dell'insieme dei
           file che devono contenere il file
           di output
8      return "ORIGINAL";
9    }
10   public String getFormatString() {
11     // Definisce la stringa di formato del
           file di output
12     return "PDF";
13   }
14   public String getDescription() {
15     // Definisce la stringa di descrizione
           del file di output
16     return "Converted␣PDF";
17   }
18   public InputStream getDestinationStream(
         InputStream source)
19           throws Exception {
20     // Crea il file di output
21     ...
22   }
23   public void postProcessBitstream(Context
         c, Item item, Bitstream
         generatedBitstream) throws Exception
         {
24     // Completa l'operazione
25     ...
26   }
27 }
```

È obbligatorio implementare questi metodi per definire completamente il *mediafilter*.

3 Implementazione dei metodi base

I due metodi principali della classe creata sono `getDestinationStream()` e `postProcessBitstream()` e la loro implementazione non è riportata nel precedente listato. Tutti gli altri metodi sono molto semplici, sono stati completamente espressi e risultano essere autoesplicativi.

L'implementazione di `getDestinationStream()` include tutto ciò che è necessario mettere in atto per realizzare il compito più semplice: compilare il codice sorgente `.tex` presente come *bitstream* nella scheda di archiviazione.

L'implementazione è la seguente:

```
1  public InputStream getDestinationStream(
       InputStream source) throws Exception {
2
3      // Crea il file tex da compilare
4      File tmpfile = createTempLaTeXFile(
           source);
5      String inputFilename = tmpfile.toPath
           ().toString();
6
7      // Esegui PDFlatex sul file .tex
8      String cmd = "pdflatex␣-output-format=
           pdf␣" + inputFilename ;
9      try {
10       String line;
11       Runtime rt = Runtime.getRuntime();
12       Process pr = rt.exec(cmd);
13     }
14     catch (Exception e) {
15       System.out.println("Exception␣in␣
             latex␣compilation");
16     }
17
18     // Raccogli il risultato
19     String output = inputFilename.
           replaceFirst(".tex\$",".pdf");
20     File fhi = new File(output);
21     byte[] textBytes = Files.readAllBytes(
           fhi.toPath());
22     ByteArrayInputStream destination = new
           ByteArrayInputStream(textBytes);
23
24     return destination;
25 }
```

Il metodo è logicamente diviso in tre parti:

1. Estrazione del file latex e salvataggio su un file temporaneo (linee 4 e 5);
2. Compilazione LaTeX (linee dalla 8 alla 16);
3. Inserimento del file risultato PDF e nel bitstream di output (linee dalla 19 alla 23).

L'unica funzione esterna usata da questo pezzo di codice, che non è presente nelle API del linguaggio Java, è `createTempLaTeXFile()`, la cui implementazione è riportata in appendice all'articolo.

Il resto del codice sorgente è molto semplice: una volta ottenuto il nome del file temporaneo creato dalla funzione `createTempLaTeXFile` nella variabile `inputFilename` lo si usa per creare la linea di comando `cmd` che è messa in esecuzione attraverso l'oggetto `Runtime` di Java (Oracle) all'interno di una sezione `try...catch`. In caso di errore del `Runtime`, questo programma prototipale si limita semplicisticamente a stampare un messaggio sullo schermo ed uscire senza creare il file.

Se invece l'operazione va a buon fine, nell'ultima parte del metodo, una volta ottenuto il nome del file di output sostituendo all'estensione `.tex` quella `.pdf`, si legge l'intero contenuto in un array di bytes che, depositati in un `ByteArrayInputStream`, sono restituiti dalla funzione.

Per concludere l'operazione è necessario compiere un *commit* nel contesto di esecuzione dell'applicazione. Questo viene eseguito nel metodo `postProcessBitstream` che ha accesso all'oggetto `Context` (Diggory (2014)):

```
public void postProcessBitstream(Context
    c, Item item, Bitstream
    generatedBitstream) throws Exception
    {
  c.commit()
}
```

Pur nella sua semplicità il programma risolve completamente il problema posto e permette la realizzazione del file PDF a partire da un file LaTeX contenuto nell'archivio. Le istruzioni necessarie alla compilazione e all'installazione del *mediafilter* sono riportate nella seconda appendice al programma.

A partire da questa base è possibile migliorare il programma in vari modi. Innanzitutto sarebbe necessario rendere più generico il programma attraverso l'uso di variabili configurabili invece che con l'utilizzo di stringhe fisse, eventualmente leggendo le variabili dal file di configurazione di `DSpace`. È possibile anche definire processi di compilazione più sofisticati che includano passi per l'inclusione delle bibliografie ad esempio o, sempre attraverso apposita configurazione, permettere l'uso di differenti processori oltre a `pdflatex`.

In parte queste alternative possono effettivamente essere espresse dal file di configurazione di `DSpace`, ma in parte forse maggiore dovrebbero adattarsi alla pubblicazione stessa connessa alla scheda. Quindi la configurazione del processo di compilazione o alcune caratteristiche dell'impaginazione potrebbero essere riportate in modo opportuno dalla scheda archivistica stessa, usando i campi appositi dello standard Dublin Core oppure adottando uno schema ad hoc proprio per questo.

È dunque pensabile integrare le informazioni della scheda archivistica (metadati):

1. per guidare la compilazione,
2. o all'interno della pubblicazione stessa.

4 Estrazione dei metadati

Per accedere ai metadati della scheda archivistica di un *elemento* è necessario interrogare l'oggetto `Item` con il metodo `getMetadata()`, i cui argomenti sono:

- nome dello schema (es. `dc`)
- nome dell'elemento (es. `contributor`)
- nome del qualificatore (es. `author`)
- la lingua dell'elemento (es. `it_IT`)

Ciascuno di questi argomenti può essere sostituito da `Item.ANY` o, limitatamente al qualificatore e alla lingua, dall'equivalente valore `null`.

Dall'esecuzione della funzione:

```
Metadatum[] values = item.getMetadata(
    schema, element, qualifer, lang);
```

si ottiene un array di metadati (classe `Metadatum[]`) che si possono usare all'interno del programma, sia per guidare la compilazione attraverso le opzioni sulla linea di comando, che per far parte del contenuto stesso (Diggory (2014)).

```
String schema = DCValue.schema
String element = DCValue.element
String qualifier = DCValue.qualifier
String lang = DCValue.language
String value = DCValue.value
```

Per comporre i valori ottenuti dai metadati all'interno del file LaTeX sono possibili due approcci alternativi:

1. Scrivere un file aggiuntivo nella directory temporanea con le necessarie informazioni, e questo implica che nel file LaTeX sorgente originale si sia introdotta un'istruzione di `\input` con il nome del file così creato (nell'esempio questo nome è `metadata.tex`).

2. Introdurre stringhe specifiche (ad esempio `$dc@contributor@author$`) direttamente all'interno del sorgente LaTeX e poi usare una libreria Java di text-templating (come Apache Velocity o FreeMarker) o i servizi delle API Java standard (come `java.text.MessageFormat` o `StringTemplate` se si vuole fare a meno di librerie aggiuntive).

Un esempio della prima linea d'azione può essere implementato con tre funzioni: una che estrae tutti i metadati dalla scheda archivistica, la seconda che prepara il contenuto del file di supporto LaTeX nel formato adeguato e la terza che scrive il file temporaneo.

```
public Map<String, List<String>>
    getMetadataHash(Item item) {
Map<String, List<String> > md = new
    HashMap<>();
Metadatum[] dcValues = item.getMetadata(
    Item.ANY, Item.ANY,
                    Item.ANY, Item.ANY);
Set<String> schemas = new HashSet<String
    >();
for (Metadatum dcValue : dcValues) {
        schemas.add(dcValue.schema);
    }
for (String schema : schemas) {
    Metadatum[] dcorevalues = item.
        getMetadata(schema,
                        Item.ANY, Item.ANY
                            , Item.ANY);
    String dateIssued = null;
    String dateAccessioned = null;
    for (Metadatum dcv : dcorevalues) {
    String qualifier = dcv.qualifier;
    if (qualifier == null) {
        qualifier = "";
    } else {
        qualifier = "@" + qualifier;
```

```
    }
    String key = schema + "@" + dcv.
        element + qualifier;
    List<String> vlist = new ArrayList<
        String>();
    if ( md.containsKey(key) ) {
        vlist = md.get(key);
    }
    vlist.add(dcv.value);
    md.put(key, vlist);
    }
}
return md;
}
```

In questa prima funzione il nome delle variabili viene costruito con il carattere `@` come separatore, ovvero così:

```
\newcommand{
  \schema@elemento@qualificatore@X
  }{ <valore> }
```

Poiché per ogni nome di metadato in `DSpace` sono sempre possibili più valori si è introdotto anche un elemento finale, indicato come `X` nel precedente esempio, ovvero una lettera (ad iniziare da `A`) che si incrementa per ogni valore. Quindi, in effetti, il primo valore della chiave `dc.contributing.author` sarà utilizzabile effettivamente nel sorgente LaTeX come `\dc@contributing@author@A`.

La funzione `getMetadataHash` restituisce una lista di chiavi-valori (possibilmente multipli) che la successiva funzione `getMetadataAsString` trasforma in un blocco di testo:

```
public String getMetadataAsString(Map<
    String, List<String>> md) {
Iterator<String> keySetIterator = md.
    keySet().iterator();
List<String> lines = new ArrayList<String
    >();
while(keySetIterator.hasNext()) {
        String key = keySetIterator.next()
            ;
        List<String> values = md.get(key);
        int index = 0;
        for (String value : values) {
          String letter = (char) (65+index
              ++);
          lines.add("\\newcommand{" + key
              + "@" + letter + "}{");
          lines.add(latex_escape(value)+"}
              ");
        }
}
String block = "% LATEX BLOCK\n\\
    makeatletter\n";
for(char line: lines) {
    block = block + line + "\n";
}
block = block + "\\makeatother\n% END
    LATEX BLOCK\n\n\n";
return block;
}
```

In questa implementazione prototipale non si è tenuto conto che le stringhe contenute nei metadati potrebbero contenere caratteri non permessi all'interno del sorgente LaTeX. Per realizzare un programma effettivamente utilizzabile bisognerebbe prevedere la trasformazione di questi caratteri non accettabile in combinazioni di caratteri accettabili da LaTeX.

La funzione che scrive il file LaTeX nella directory temporanea è:

```
private void createTempFile(String dir,
    Item item) throws IOException {
    Map<String, List<String>> md =
        getMetadataHash(item);
    String metadataStr =
        getMetadataAsString(md);
    FileUtils.writeStringToFile(new File(
        dir + "metadata.tex"), metadataStr
        );
}
```

All'interno del file sorgente LaTeX iniziale è quindi necessario introdurre nel preambolo l'istruzione:

```
\input{metadata}
```

e poi utilizzare nel testo i comandi creati dalle variabili.

L'oggetto `Item` non è presente tra i parametri del metodo `getDestinationStream()`, quindi tutte le operazioni necessarie all'acquisizione dei metadati devono essere compiute nell'implementazione di un ulteriore metodo denominato `preProcessBitstream()` che, come dice il nome, viene eseguito prima della funzione di elaborazione del bitstream (cioè `getDestinationStream()`)

```
    public boolean preProcessBitstream(
        Context c, Item item, Bitstream
        source) throws Exception {
        // Metodo eseguito prima di
            realizzare la conversione, puo'
            servire ad estrarre i metadati
            utili
        tempPath = Files.
            createTempDirectory("latex");
        createTempFile(tempPath.toString(),
            item);
    }
```

Allo stesso modo, qualora sia necessario modificare il valore dei metadati o scriverne di nuovi (ad esempio per salvare informazioni sull'andamento della compilazione), è necessario interagire con l'oggetto `Context` di `DSpace` che non è presente nel metodo `getDestinationStrem()` (Diggory (2014)). Anche in questo caso può usarsi il metodo `preProcessBitstream` o, forse più adeguato, il metodo `postProcessBitstream` che viene messo in esecuzione solo se l'operazione di trasformazione è andata a buon fine.

5 Conclusioni

Pur nella sua estrema sintesi e semplificazione, quest'articolo presenta un esempio completo di realizzazione di un *mediafilter* per `DSpace`. Il programma è utile ad organizzare collezioni di documenti presenti nel repository direttamente in formato sorgente, lasciando a `DSpace` il compito di creare i file PDF come sottoprodotto.

Le possibilità di estensione di quest'esempio sono numerose e solo per citarne alcune è possibile considerare i seguenti scenari:

1. Spesso una pubblicazione è composta da più di un file `.tex`, quindi potrebbe essere necessario caricare nell'archivio differenti file sorgente. In questo caso le strategie possibili possono essere due.

   (a) Usare un unico file compresso (come un file `.tar` o `.zip`) contenente tutti i sorgenti necessari da scompattare prima della compilazione.

   (b) Caricare singolarmente tutti i file necessari nella scheda archivistica insieme ad un *file di produzione* che racchiuda i nomi dei bitstream da coinvolgere nella produzione del prodotto. Questo file di produzione potrebbe avere un'estensione ad hoc (ad esempio `task`) che permetterà di mettere in esecuzione il *mediafilter*.

2. Le pubblicazioni di natura scientifica o statistica potrebbero prevedere un aggiornamento automatico dei dati sottostanti. Per realizzare un tale automatismo si può registrare un processo periodico di cancellazione dei file PDF prodotti e adottare all'interno del file LATEX sistemi di acquisizione dei dati dalle fonti utilizzando estensioni del linguaggio LATEX come `pythontex` (Poore, 2015) o considerando l'introduzione di elaborazioni nei linguaggi di programmazione scientifica come passi propedeutici alla compilazione dei documenti.

3. Piuttosto che adottare LATEX come file sorgente è sempre possibile usare un formato più astratto, dal più semplice `Markdown` (Ovadia (2014)) al più sofisticato `Docbook` (Martínez-Ortiz *et al.* (2006)), che possa generare differenti tipi di file prodotto (PDF, HTML, ePub, ecc.). In questo caso si potrà utilizzare un processore generico come `pandoc` (Dominici (2014)) o `expand` (Krijnen *et al.* (2014)) per compiere il lavoro di compilazione.

Queste e altre simili estensioni possono far diventare un'istanza `DSpace` non solo un repository statico di documenti, ma un processore di flussi di lavoro il cui risultato, pronto per essere consultato dall'utente finale, viene conservato all'interno delle schede archivistiche.

Appendice 1: La funzione `createTempLaTeXFile()`

Per completezza espositiva si riporta in quest'appendice il codice sorgente del metodo `createTempLaTeXFile()` che peraltro non presenta caratteri di particolare complessità.

```
1  private File createTempLaTeXFile(
       InputStream source) throws IOException
        {
2      File f = File.createTempFile("document
           ", ".tex");
3      f.deleteOnExit();
4      FileOutputStream fos = new
           FileOutputStream(f);
5      byte[] buffer = new byte[1024];
6      int len = source.read(buffer);
7      while (len != -1) {
8          fos.write(buffer, 0, len);
9          len = source.read(buffer);
10     }
11     fos.close();
12     return f;
13 }
```

L'unica annotazione da fare su quest'implementazione è che nel caso del sorgente riportato a pag. 4 sarà necessario aggiungere alla creazione del file anche il parametro relativo alla directory temporanea in cui viene posto il file dei metadati. Quindi la riga 2 diventerebbe:

```
1      File f = File.createTempFile("document
           ", ".tex",tempPath);
```

Il codice sorgente dei programmi riportati nell'articolo è disponibile all'indirizzo `https://github.com/exedre/2018-arstexnica-latex-dspace`.

Appendice 2: Compilazione del *mediafilter*

Per integrare il *mediafilter* realizzato all'interno di una installazione di `DSpace` è necessario procedere alla sua compilazione all'interno del codice sorgente di `DSpace`.

Il file Java contenente la classe del nuovo *mediafilter* va posizionata all'interno della directory

```
1 cp mediafilter.java [dspace-src]/dspace-
      api/src/main/java/org/dspace/app/
      mediafilter
```

per poi eseguire il comando di compilazione:

```
1 cd [dspace-src] && mvn package
```

Al termine della compilazione si potrà installare il nuovo filtro con il comando:

```
1 cd [dspace-src]/dspace/target/dspace-
      installer/ && ant update
```

Una volta installato il nuovo *mediafilter* sarà necessario riportarlo nella configurazione aggiungendo un nome nella chiave di configurazione `filter.plugin`:

```
1 filter.plugins = <Altri nomi di filtri>,
      Nuovo Mediafilter
```

e aggiungendo la classe del filtro nell'elenco dei filtri di formato:

```
1 plugin.named.org.dspace.app.mediafilter.
      FormatFilter = \
2   org.dspace.app.mediafilter.
        nuovoMediafilter = Metadata Exporter
        , \
3   [...]
```

Infine è necessario indicare il tipo o i tipi dei file di input che dovranno mettere in esecuzione il filtro indicato:

```
filter.org.dspace.app.mediafilter.
    nuovoMediafilter.inputFormats =
    Markdown
```

Riferimenti bibliografici

De Robbio, A. (2002). «Relazione tecnica su dspace (mit)».

Diggory, M. (2014). «Business logic layer». URL https://wiki.duraspace.org/display/DSDOC5x/Business+Logic+Layer.

Dominici, M. (2014). «An overview of pandoc». *TUGboat*, **35** (1), pp. 44–50.

Donohue, T. (2016). «Mediafilters for transforming dspace content». URL https://wiki.duraspace.org/display/DSDOC5x/Mediafilters+for+Transforming+DSpace+Content.

Krijnen, J., Swierstra, D. e Viera, M. O. (2014). «Expand: Towards an extensible pandoc system». In *International Symposium on Practical Aspects of Declarative Languages*. Springer, pp. 200–215.

Martínez-Ortiz, I., Moreno-Ger, P., Sierra, J. L. e Fernández-Manjón, B. (2006). «Using docbook and xml technologies to create adaptive learning content in technical domains.» *IJCSA*, **3** (2), pp. 91–108.

Oracle. «Class runtime». URL https://docs.oracle.com/javase/7/docs/api/java/lang/Runtime.html.

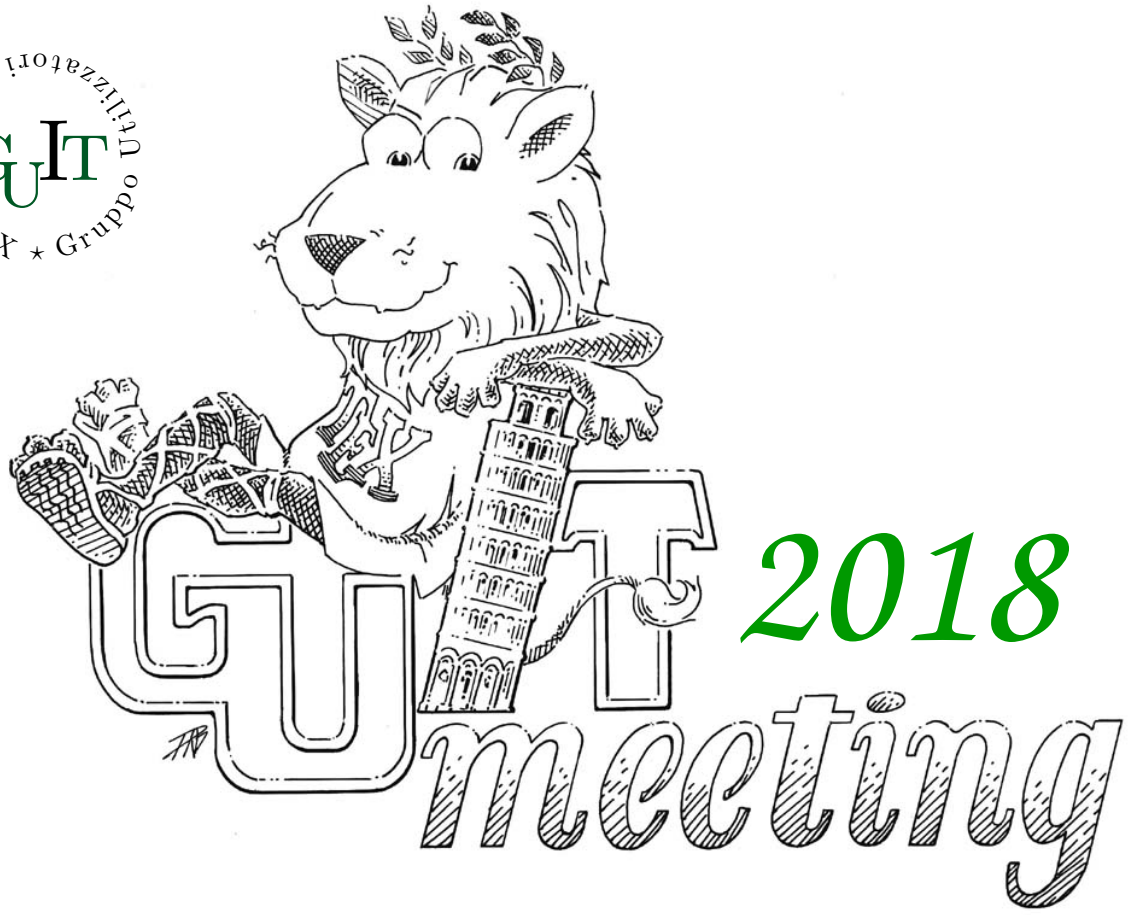
Ovadia, S. (2014). «Markdown for librarians and academics». *Behavioral & Social Sciences Librarian*, **33** (2), pp. 120–124.

Poore, G. M. (2015). «Pythontex: reproducible documents with latex, python, and more». *Computational Science & Discovery*, **8** (1), p. 014 010.

▷ Emmanuele Somma
Biblioteca Paolo Baffi
Servizio Struttura Economica
Dipartimento Economia e Statistica
Banca d'Italia
emmanuele dot somma at bancaditalia dot it

Questa rivista è stata prodotta
dal Gruppo Utilizzatori Italiani di TEX
usando esclusivamente software libero.

Versione elettronica per la diffusione via web.



Quindicesimo convegno nazionale su TEX, LATEX e tipografia digitale

Roma, 20 ottobre 2018

Call for Papers

Sabato 20 ottobre a Roma si terrà il quindicesimo Convegno annuale su TEX, LATEX e tipografia digitale organizzato dal Gruppo Utilizzatori Italiani di TEX. Il Convegno sarà un momento di ritrovo e di confronto per la comunità LATEX italiana, tramite una serie di interventi atti sia a contribuire all'arricchimento sia a supportarne lo sviluppo.
Maggiori informazioni sul Convegno e sulle modalità di presentazione degli interventi saranno disponibili all'indirizzo:

`http://www.guitex.org/guitmeeting/2018/`

ArsTEXnica

Rivista italiana di TEX e LATEX

Numero 25, Aprile 2018

9 771828 236001 80025