Numero 26
Ottobre
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 Ottobre 2018

GuIT meeting 2018

QUINDICESIMO CONVEGNO NAZIONALE SU TEX, LATEX E TIPOGRAFIA DIGITALE

20 ottobre 2018

Sapienza Università di Roma

Sala degli Affreschi della facoltà di Ingegneria,
via Eudossiana 18, Roma

Programma del convegno

9:30 Benvenuto. Inizio dei lavori.

9:40 *Cenni sulla produzione di ebook con LATEX e Calibre*. Gianluca Pignalberi.
10:10 *Aggiornamento di* arara. Claudio Beccari.
10:20 *Introduzione al pacchetto* `xparse`. Claudio Beccari.

10:50 — Pausa caffè (40 min).

11:30 *Esperienze nella pubblicazione di un volume di atti di convegno*. Massimiliano Dominici.
12:00 *Axessibility: creating PDF documents with accessible formulae*. D. Ahmetovic, T. Armano, M. Berra, C. Bernareggi, A. Capietto, S. Coriasco, N. Murru e A. Ruighi.

12:30 — Pausa pranzo (1 h 30 min).

14:00 *Experimenting with* `makeindex` *and Unicode, and deriving* `kameindex`. Antoine Bossard e Keiichi Kaneko.
14:30 *Connecting LuaTEX to MongoDB*. Roberto Giacomelli.
15:00 *Come postare correttamente sul Forum del GuIT e vivere felici*. Herr Professor Paulinho van Duck.

15:30 Chiusura lavori.

16:00 Riunione annuale del gruppo.

17:30 Chiusura dei lavori. Arrivederci.

La partecipazione è libera e gratuita, previa registrazione entro il 15 ottobre.
Registrazione online: www.guitex.org/home/meeting

GuIT · Gruppo Utilizzatori Italiani di TEX

www.guitex.org

SAPIENZA UNIVERSITÀ DI ROMA

ArsTEXnica

Rivista italiana di TEX e LATEX

Numero 26, Ottobre 2018

Gruppo Utilizzatori Italiani di TEX

Editoriale

Claudio Beccari

Questo numero di ArsTEXnica contiene, oltre agli atti del GUIT *meeting 2018* del 2018, anche la versione stampata del numero di aprile, già presente come file PDF nel sito del GUIT.

Questo Meeting di Roma è importante perché nella prima metà del 2018 sono successe molte cose, alcune delle quali impattano fortemente sul funzionamento della nostra associazione.

Per fortuna, però, non influiscono sulla nostra volontà di incontrarci, di discuterne e, specialmente, di presentare i nostri lavori, alcuni dei quali sono fortemente innovativi.

Gli articoli di questo numero di ArsTEXnica sono qui elencati in ordine sparso, indipendentemente da come sono esposti oralmente durante il Meeting.

I due autori Antoine Bossard e Keiichi Kaneko scrivono dal Giappone; usando molto LaTeX in un ambiente nel quale si usano solo caratteri Unicode per gestire i vari sistemi di scrittura giapponesi, si trovano nella necessità di comporre correttamente gli indici analitici; hanno perciò sviluppato un fork di makeindex, chiamato kameindex, che permette di gestire correttamente le voci da indicizzare scritte con qualsiasi alfabeto. L'articolo è chiaro e comprensibile anche ai non addetti ai lavori.

Massimiliano Dominici ci racconta la sua esperienza per impaginare con il sistema TeX un libro di atti di un congresso dedicato a Pierre Souffrin, storico della scienza. Gli articoli di questi atti sono scritti in italiano, inglese e francese; riportano molti disegni e figure sia di origine leonardiana, sia predisposti dagli oratori di quel congresso; ma gli articoli arrivati al curatore erano stati composti dai loro autori usando i più svariati sistemi di videoscrittura. Il tutto andava omogeneizzato: molti disegni andavano rifatti, i font dovevano essere scelti con cura e in modo che disponessero di tutti i glifi necessari. L'articolo di Dominici descrive con cura il lavoro che c'è dietro ad una pubblicazione del genere; oggi forse esisterebbero altre soluzioni rispetto a quelle usate allora, ma è molto ben descritto il lavoro delicato necessario per realizzare pubblicazioni collettanee.

Nell'ambito della sua professione Roberto Giacomelli deve comporre documenti di vario genere elaborando informazioni che si trovano sparse in file di diversa origine, ma che contengono dati da elaborare ulteriormente. Giacomelli ci ha già presentato diversi articoli per mostrare come Lua(La)TeX sia fondamentale per gestire il suo flusso di lavoro. In questo particolare articolo ci mostra come usare Lua(La)TeX per interrogare un particolare database al fine di prelevare dati e informazioni varie da elaborare ulteriormente per inserirne il contenuto informativo nel documento composto. In particolare questo articolo descrive come interconnettere Lua(La)TeX ad un database residente su un server MongoDB; MongoDB gestisce database di tipo NoSQL, cosa che è particolarmente comoda per interagire con il linguaggio Lua.

Van Duck è lo pseudonimo di un collaboratore regolare di *TUGboat* che in modo spiritoso scrive indicazioni per i lettori quando si usa il sistema TeX. In questo articolo ci parla di cosa fare quando gli utenti incontrano degli errori o hanno delle difficoltà di composizione: la soluzione spesso richiede la creazione di un file che costituisca un efficace esempio minimo compilabile da allegare alla richiesta di aiuto nei vari siti dove gli utenti possono farlo; fra questi siti, ovviamente, c'è anche il Forum del GUIT. L'articolo, già apparso in inglese su *TUGboat*, non è semplicemente tradotto ma è adattato alla situazione dell'utente italiano.

Gianluca Pignalberi ci descrive la sua esperienza nel trasformare un testo composto con LaTeX in formato EPUB. o meglio, ci descrive come produrre un ebook in formato EPUB usando gli strumenti del sistema TeX oltre ad altri strumenti per la conversione dei formati dei file. Per una casa editrice che riceva un testo da pubblicare a stampa, il lavoro di impaginazione è abbastanza standardizzato e ricorre a programmi oramai stabili. Il lavoro per trasformare lo stesso testo in formato EPUB, specialmente se con contenuti tecnico-scientifici, ricchi di formule, figure, tabelle, e altre simili costruzioni tipografiche, diventa un lavoro manuale tremendamente laborioso. Dalla sua esperienza, raccontata in dettaglio in questo articolo, si imparano molte cose da fare o da evitare.

Claudio Beccari presenta una specie di tutorial sull'uso del pacchetto xparse che consente di definire delle funzioni (il nome delle macro usato dal linguaggio L3) che possono accettare argomenti di tipo booleano e di tipo con valori assegnati, obbligatori e facoltativi, variamente delimitati – per quelli facoltativi, se devono essere dotati di un valore di default. Le possibilità sono molto numerose e permettono di definire funzioni in modo decisamente più semplice che non con i comandi nativi di TeX e LaTeX. I diversi esempi presentati dimostrano la flessibilità delle funzioni definite con xparse.

Il gruppo di lavoro che lavora presso il dipartimento di matematica dell'università di Torino

ha predisposto un interessantissimo contributo a molte mani; gli autori sono Dragan Ahmetovic, Tiziana Armano, Michele Berra, Cristian Bernareggi, Anna Capietto, Sandro Coriasco, Nadir Murru, Alice Ruighi. L'articolo prosegue, in un certo senso, il lavoro che il gruppo presentò al Meeting di Brescia del 2016, in merito all'uso di LaTeX per comporre testi in PDF. In questo articolo raccontano come siano riusciti a tradurre in parole anche la matematica composta con LaTeX mediante opportune interfacce normalmente usate per la lettura del testo. I problemi sono moltissimi e la meta finale sarà la normativa ISO relativa ai file "universalmente accessibili" (PDF/a-1ua) a cui stanno lavorando molti gruppi in diverse nazioni oltre che presso l'International Standards Organisation. Il software da loro prodotto è già disponibile in TeX Live 2018; quindi chiunque sia interessato a sostenere la disabilità degli ipovedenti ha già uno strumento a disposizione per comunicare loro anche la matematica con le sue formule.

Il numero termina con una breve comunicazione relativa ad un articolo comparso nel precedente numero di ArsTEXnica dove si descriveva un "Uso insolito di **arara**". Quando quell'articolo era stato scritto, non era ancora disponibile la documentazione di **arara** v.4.0, quindi gli esempi d'uso erano stati scritti usando la sintassi della precedente versione di **arara**. Era stato promesso che non appena fosse stata pubblicata e resa disponibile su CTAN la nuova versione sarebbero stati pubblicati i codici aggiornati. Eccoli qui.

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`

Cenni sulla produzione di ebook con LaTeX e Calibre

Gianluca Pignalberi

Sommario

La produzione di libri elettronici (ebook) parallela alla produzione di libri cartacei sembra essere ormai la norma. Vediamo una possibile via di trasformazione dei documenti LaTeX in file EPUB standard.

Abstract

Making ebooks along with paper books seems to be normal now. We will see a possible way to transform LaTeX documents into standard EPUB files.

1 Introduzione

Nel mio vecchio articolo Pignalberi (2015) accennavo al fatto che una parte del mio lavoro di compositore LaTeX è assorbita dalla produzione di ebook in formato EPUB dei libri di cui realizzo i PDF, siano essi destinati alla sola distribuzione in formato elettronico o anche alla stampa. Per il mio lavoro ricorro esclusivamente al software libero: LaTeX per i libri e, se possibile, le copertine; GIMP per il fotoritocco (col grosso handicap della gestione CMYK); Calibre per la conversione da HTML a EPUB; Gnuplot per i grafici; QTi*k*Z per i disegni in Ti*k*Z; una serie di altri programmi più o meno noti per altre necessità di lavorazione. A questi strumenti devo aggiungerne due: Sigil (Schember, 2018), un potente editor di ebook che integrava il vecchio programma standard per la validazione dei documenti EPUB (FlightCrew), e `epubcheck` (Epubcheck team, 2018), attuale validatore standard, che non è rilasciato sotto licenza GPL.

Nell'articolo spiegherò le varie fasi per ottenere un EPUB conforme allo standard e, dunque, accettato dalle più note e comuni piattaforme di vendita di ebook, a partire da un documento LaTeX.[1]

Dietro suggerimento del direttore ho usato un documento scientifico invece che uno dei saggi letterari che produco di solito. Il documento è costituito da un capitolo del libro D'Antona (1998) e ci permetterà di vedere formule e grafici, ma non testo in greco o in ebraico che genera problemi diversi. Riprodussi quasi esattamente[2] tale capitolo nel 2013 insieme ai miei allievi del terzo (e ultimo) anno del corso di Operatore Grafico Multimediale (la mia materia era denominata Open Source) presso Latina Formazione Lavoro. Lo scopo era addestrarli all'uso di LyX e di Ti*k*Z (il primo era un tentativo di far digerire loro gli accenni a LaTeX mascherando quest'ultimo da editor WYSIWYG, il secondo era parte del programma sul disegno vettoriale) e per mostrare loro con quanta facilità ottenevamo testi matematici di aspetto migliore di quelli ottenibili con i programmi di composizione tipografica studiati in altre materie. Possiamo vedere alcune delle pagine prodotte nella figura 1. Di tale documento producemmo anche la versione EPUB, non validata. La validazione verrà fatta appositamente per questo articolo in virtù della sua finalità di mostrare come creare un prodotto vendibile.

Naturalmente quello mostrato è un progetto didattico, ottimo per studiare alcuni aspetti della lavorazione. Nei limiti della mia esperienza parlerò anche di altri problemi legati alla produzione di ebook. Il progetto didattico è troppo limitato per mostrare tutti i problemi affrontati per lavoro.

2 Primo tempo: da LaTeX a HTML

Lo strumento che uso per compilare un documento LaTeX e ottenere un file HTML è `htlatex` (Gurari, 2004). Pur essendo per costruzione meno flessibile di LaTeX (non è progettato per usare una classe il cui nome non sia uno di quelli standard di LaTeX — book, report, article, letter e pochissimi altri — né pacchetti di cui non sia stato scritto il "porting"), mi permette con (relativamente) poco lavoro di ottenere un HTML a cui serviranno solo alcune modifiche. Naturalmente `htlatex` non è l'unico strumento utilizzabile. Ritengo che `pandoc` potrebbe essere addirittura migliore se veramente mi permettesse di 1) saltare il passaggio intermedio della mia catena di lavoro (LaTeX → HTML → EPUB) e 2) lasciare intatti i miei documenti senza le modifiche necessarie, anche solo al preambolo, per ovviare alle limitazioni e ai vincoli di `htlatex`.

Esiste anche un pacchetto per XeLaTeX e LuaLaTeX (Hoftich, 2016, il refuso nel titolo è quello originale) che simula il risultato di `htlatex` ma io l'ho sperimentato pochissimo e senza alcun successo. Tra l'altro il progetto, iniziato almeno nel 2013, ha avuto un ultimo commit nel 2016 e non sembra essere più attivo.

Come detto in precedenza, `htlatex` non è un prodotto flessibilissimo, anche se alcune libertà ce le permette. Per comporre i libri per lavoro sono

1. Tutte le prove descritte nell'articolo sono state ottenute con i programmi inclusi in TeX Live 2017.

2. Poiché il progetto con gli allievi non prevedeva la riproduzione *esatta* dell'aspetto del libro, alcuni elementi nelle pagine degli esercizi non sono posizionati al millimetro, sebbene la cosa potrebbe non balzare immediatamente a occhi meno esperti.

4 I grafi cordali e la Ricorrenza Master

4.1 Il polinomio cromatico come combinazione lineare di fattoriali decrescenti

Come abbiamo avuto occasione di dire (ben più di una volta) nelle pagine precedenti, il polinomio cromatico di un grafo indica il numero di modi in cui lo si può colorare usando *al più t* colori. Più precisamente sappiamo che, per un valore di *t* arbitrariamente fissato, $p(G;t)$ sarà il numero di modi di colorare G con uno, due o t colori. Ad esempio le due funzioni $crom_1, crom_2 : \{v_1, v_2\} \to \{BLU, GIALLO, ROSSO, VERDE\}$ tali che

$$crom_1(v_1) = ROSSO = crom_1(v_2)$$

$$crom_2(v_1) = VERDE, crom_2(v_2) = BLU$$

sono due delle 16 colorazioni di I_2 che si possono realizzare avendo a disposozione quattro colori (infatti $p(I_2; 4) = 4^2 = 16$) anche se abbiamo effettivamente utilizzato prima un solo colore e poi soltanto due. In realtà la cosa non ci sorprende più di tanto per il buon motivo che alle funzioni che rappresentano le colorazioni non è richiesto di essere surriettive. Quindi, se fossimo interessati ad una classificazione delle colorazioni di un grafo in termini di *quanti* colori vengono effettivamente usati, e non tanto di *quali*, potremmo sospettare che il polinomio cromatico (almeno così come lo abbiamo presentato fin'ora) non sia lo strumento più adatto. Definiamo allora un nuovo coefficiente relativo alle colorazioni di un grafo: con la notazione $E_{n,k}(G)$ indicheremo *il numero di colorazioni di un grafo G di ordine n che utilizzano effettivamente k colori* ([1]). In tal modo abbiamo imposto una sorta di surrettività a queste colorazioni che potremmo quindi chiamare *k-suriettive*, o anche *colorazioni esatte*.

Per capire bene il concetto, commentiamone alcune proprietà generali. In primo luogo risulta $E_{n,m}(G) = 0$ per ogni $m > n$. E fin qui tutto bene. Poi si ha sempre $E_{n,n}(G) = 1$.

[1] Occasionalmente, ove non ci sia rischio di confusione ci limiteremo a scrivere $E_{n,k}$ in luogo di $E_{n,k}(G)$

Ogni grafo cordale è ottenuto con una serie di incastri piramidali a partire da I_0. Ma si badi bene: dopo aver effettuato un'operazione di piramide, il grafo risultante conterrà una cricca di ordine 1 o maggiore. Inoltre è ovvio che la più grande cricca che un grafo di ordine n può contenere è proprio K_n. Anzi, a proposito dei grafi completi, possiamo dire che questi sono ottenuti con una serie di incastri piramidali massimali poiché, ad ogni passo, il grado del nuovo vertice è il più grande possibile. In altre parole, K_n è ottenuto aggiungendo ad I_1 delle piramidi di grado $1, 2, \ldots, n-1$. All'altro estremo si trovano proprio i grafi I_n che sono invece ottenuti a partire da ad I_1 con una serie di incastri piramidali minimali (anzi minimi) cioè di grado 0. Dunque la sequenza dei gradi delle piramidi con cui abbiamo costruito I_n è $0, 0, \ldots, 0$. Questa caratterizzazione ha il gradito effetto collaterale di farci conoscere la struttura del polinomio cromatico di tutti i grafi cordali. Infatti basta osservare che i polinomi cromatici di K_n e di I_n sono rispettivamente, $(t)_n$ e t^n per iniziare a sospettare che detti $r_1, r_2, r_3, \ldots, r_n$ i gradi delle piramidi con cui è stato costruito il generico grafo cordale G di ordine n, il suo polinomio cromatico sarà

$$p(G;t) = (t - r_1)(t - r_2)(t - r_3)\ldots(t - r_n),$$

dove $0 \leq r_1 \leq r_2 \leq \ldots \leq r_n \leq n - 1$. E in effetti è così. Abbiamo dunque conseguito il semplice, ma importante risultato che recita: *tutte le radici del polinomio cromatico di un grafo cordale sono intere.* Ma attenzione: è della massima importanza tener presente che

IL CONTRARIO NON È VERO ! ! !

Nel prossimo capitolo analizzeremo diffusamente questa affermazione, ma ora limitiamoci ad osservare che, se spogliamo la scrittura del polinomio cromatico di un grafo in termini di fattoriali decrescenti dalla sua veste pittorica, ciò che resta è una semplice relazione algebrica tra polinomi. In altre parole, l'espressione

$$p(G;x) = \sum_{k=0}^{n} E_{n,k}(G)(x)_k$$

può essere pensata come l'espressione del polinomio $p_n(x) = p(G;x)$ come combinazione lineare di fattoriali decrescenti in cui le costanti di connessione sono indicate da $E_{n,k}(G)$ in luogo dell'usuale (almeno per noi) simbolo $L_{n,k}$. A questo punto si può osservare che i polinomi cromatici dei grafi che costituiscono una sequenza di eliminazione di un grafo cordale sono una famiglia persistente di polinomi. Dunque la ricorrenza master diventa una relazione tra le colorazioni esatte di un grafo cordale e quelle del grafo che lo precede in una sequenza di eliminazione ([2]). Per interpretare la ricorrenza in termini di grafi, suddividiamo in due classi tutte le partizioni indipendenti di G_n in k blocchi. Nella prima metteremo tutte quelle partizioni in cui il vertice v_n costituisce un blocco a sé stante, e nella seconda tutte le altre. Ovviamente la prima classe conterrà tanti elementi quante le partizioni indipendenti di G_{n-1} in $k-1$ blocchi. Infatti aggiungendo a ciascuna di esse un blocco contenente il solo v_n

[2] Ovvero tra le colorazioni esatte di un generico grafo H e quello ottenuto costruendo una piramide su di esso.

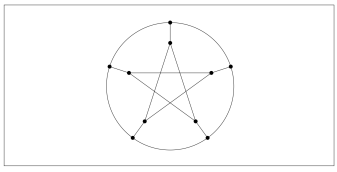


Figura 4.6: Il grafo di Petersen.

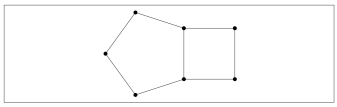


Figura 4.7: Questo grafo non è perfetto.

Tra l'altro, questo ragionamento si estende facilmente a tutti i grafi che contengono, come sottografo indotto, dei circuiti di lunghezza dispari (maggiore di 4) come le ruote W_{2m+1} con $m > 1$. Inoltre, l'elenco di famiglie di grafi perfetti può, per così dire, essere duplicato poiché si sa che *il complemento di un grafo perfetto è perfetto.* Su questo tema il famoso grafista francese C. Berge ha congetturato che *un grafo G sia perfetto se e soltanto se né G né il suo complemento contengono come sottografi indotti circuiti dispari di lunghezza maggiore di* 4.

Passiamo ora al numero di copertura che, nel folklore della teoria, viene interpretato come il minimo numero di guardiani necessari a tenere sott'occhio tutti i corridoi di un museo. Un insieme C di vertici di un grafo G è detto *copertura dei lati* di G se ogni suo lato è incidente in

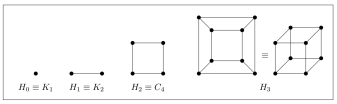


Figura 4.8: Quattro ipercubi.

almeno uno dei vertici di C. Ad esempio, tre vertici qualunque di C_4 sono una sua copertura. Il numero di vertici di una copertura di ordine minimo è detto il *numero di copertura* di G e viene indicato con $\alpha(G)$. Ad esempio, due vertici non consecutivi di C_4 sono una sua copertura, e in generale è facile vedere che $\alpha(C_n) = \left\lceil \frac{n}{2} \right\rceil$, e lo stesso vale per i cammini con n punti. se invece TT_n è un albero che non sia un cammino, allora $\alpha(TT_n) < \left\lceil \frac{n}{2} \right\rceil$. Per le ruote invece si ha $\alpha(W_n) = 1 + \left\lceil \frac{n}{2} \right\rceil$.

Il *numero di copertura in cricche* di un grafo è il minimo numero di sottografi completi necessari a ricoprire tutti i suoi vertici, e viene indicato con $\kappa(G)$. Ad esempio, $\kappa(W_3) = 1, \kappa(W_4) = 2$ e, per $n > 4$, $\kappa(W_n) = 1 + \left\lceil \frac{n}{2} \right\rceil$. Ricordiamo che si può dimostrare la seguente caratterizzazione: un grafo G è perfetto se e soltanto se per ciascuno dei suoi sottografi indotti, S - ivi incluso G stesso - risulta $\alpha(S) = \kappa(S)$. Questo risultato ci dice che, dal punto di vista dei grafi perfetti, il numero di copertura in cricche e il numero cromatico giocano lo stesso ruolo.

Infine ricordiamo che un *insieme dominante* (o *ricoprente*) di un grafo è un sottoinsieme D dei suoi vertici con la proprietà che ogni altro vertice del grafo sia adiacente ad almeno un vertice di D. Il *numero di dominanza* di un grafo è l'ordine di un insieme dominante minimo. Questo concetto è di grande rilevanza nella teoria dei codici a correzione d'errore in cui però i grafi di cui ci si occupa sono quasi esclusivamente gli ipercubi, H_n(vedi Figura 4.8). È facile vedere che il numero di dominanza di H_1, H_2, H_3 e H_4 è, rispettivamente, 1, 2, 3 e 4. Ma è ben strano che per ricoprire i 32 vertici di H_5 occorra marcare 7 vertici. Sono noti anche i numeri di dominanza H_6, H_7 e H_8che sono 12,16 e 32, ma non quelli di ipercubi più grandi. La tabella qui sotto riporta le migliori limitazioni conosciute per numeri di dominanza di $H_9, H_{10}, H_{11}, H_{12}$ e H_{13}.

Numero di vertici	Limite inferiore	Limite superiore
9	55	62
10	105	120
11	178	192
12	342	380
13	598	736

FIGURA 1: Pagine riprodotte da D'ANTONA (1998).

solito usare una versione pesantemente modificata di book (Pignalberi, 2015), corredata di plugin e rinominata in ossequio alla LPPL. Per le tabelle, invece, uso tabu per la sua flessibilità (sebbene debba poi fare dei piccoli aggiustamenti). Entrambi i software non sono compatibili con l'uso di `htlatex`. Per ovviare al primo problema mi limito a prendere una nuova classe book, modificarla molto meno pesantemente (solo per aggiungere il frontespizio, il retrofrontespizio e alcuni comandi usati frequentemente) e, senza rinominarla, compilare senza problemi per ottenere un HTML accettabile. Per far funzionare le tabelle, invece, devo tornare al "primitivo" tabular, dovendo quindi modificare la sintassi della dichiarazione delle colonne (e sovente ricorrere a `p{}` con frazioni di `\textwidth` per non uscire "fuori margine"). La nota positiva è che non devo preoccuparmi delle tabelle lunghe perché nelle pagine HTML non manca lo spazio verticale (e l'EPUB gestisce la lunghezza e la suddivisione delle tabelle come lo farebbe longtable).

Per il progetto didattico non sussiste alcuno dei suindicati problemi perché il documento è basato sulla classe book (con le modifiche fatte direttamente nel preambolo del documento) e l'unica tabella è stata composta con tabular. Nel listato 1[3] vediamo il preambolo del documento che ci permette di ottenere un documento dall'aspetto uguale a quello della figura 1.

Listato 1: Preambolo del documento riproducente il libro D'Antona (1998).

```
\documentclass{book}

\usepackage[T1]{fontenc}
%\usepackage[utf8x]{inputenc}
\usepackage[latin1]{inputenc}
\usepackage[italian]{babel}
\usepackage{color}
\usepackage{times}
%\usepackage{mathptmx}
\usepackage{tikz}
\usepackage{amsmath}
\usepackage[paperwidth=17cm,
    paperheight=24cm,outer=2cm,
    top=3.3cm,bottom=1.5cm]{
    geometry}
\usepackage[cam,a4,center]{crop}
\usepackage[format=plain,
    labelfont=it,textfont=it,
    justification=justified,
    singlelinecheck=false,skip=5
    mm]{caption}
\usepackage{fancyhdr}
\pagestyle{fancy}{%
\fancyhf{}
\fancyhead[RE]{\itshape\leftmark
    }
\fancyhead[LO]{\itshape
    \chaptername~\thechapter}
\fancyhead[LE,RO]{\itshape
    \thepage}

\makeatletter
\def\@makechapterhead#1{%
  \begin{minipage}[t]{.15\
     textwidth}
  \vskip-2pt
  {\fontsize{84}{0}\selectfont
     \textcolor{gray}{
     \thechapter}}
  \end{minipage}
  \begin{minipage}{.80\textwidth
     }
  \rule{0pt}{100pt}{\LARGE{
     \textbf{\ \ #1}

\hrulefill}}\end{minipage}
  \vskip66pt
  }
\renewcommand\section{
    \@startsection {section}{1}{
    \z@}%
    {-3.5ex \@plus -1ex \@minus
       -.2ex}%
    {2.3ex \@plus.2ex}%
    {\normalfont\large\itshape}}
\makeatother
\parindent=14pt
\parskip=0pt
\frenchspacing
```

3. Alcune righe dei listati numerati, troppo lunghe per entrare interamente nella colonna di testo, sono state interrotte automaticamente e portate a capo. Potete notare queste interruzioni dalla mancanza del numero di riga. Questo significa che non c'è alcuno spazio o invio tra un l'ultimo carattere di una riga di codice e il primo della successiva riga non numerata.

Più articolato è il discorso sulle figure. Se dobbiamo includere immagini raster non ci sono problemi: basta limitarci ai formati JPG e PNG. Già con i formati vettoriali EPS e PDF siamo pressoché costretti a una conversione e all'eventuale modifica del sorgente LATEX. Sebbene `htlatex` tenti una conversione delle immagini in questi formati, non sempre il risultato è almeno accettabile. Nel caso poi di figure contenenti il sorgente del disegno (pict2e, Ti*k*Z...), siamo costretti a generarle separatamente e a convertirne il formato, quindi a modificare il sorgente LATEX nel punto di inclusione dell'immagine. Siccome JPG e PNG non sono il massimo per riprodurre immagini vettoriali (meglio comunque il PNG), possiamo convertire questa tipologia di immagini in `.svg`, conformemente allo standard. Ho fatto una prova e il risultato è il seguente: Firefox 55.0.2 mostra le immagini SVG dell'HTML senza problemi. La conversione a ebook avviene nel modo descritto nella sezione 4 e, mentre Sigil mostra tutte le figure nel loro splendore, un lettore pur versatile come Calibre no. Dunque è

rischioso avventurarsi fuori dai due formati raster citati all'inizio del capoverso perché rischiamo che alcuni lettori commerciali non mostrino alcunché. Nel progetto didattico non ce ne discosteremo, pur usando immagini JPG a 72 dpi invece che PNG a una risoluzione leggermente maggiore:[4]

```
137 \begin{figure}
138 \includegraphics[width=
      \textwidth]{figura1.jpg}
139 \emph{\caption{$V=\{a,b,c\},\, E
      =\{(b,c)\}$\hspace{26em}}}
140 \end{figure}
```

del sorgente da dare in input a `htlatex` sostituisce

```
138 \begin{figure}
139 \framebox{
140 \begin{minipage}{.975\textwidth}
141 \vskip2mm
142 \centering
143 \begin{tikzpicture} \coordinate
      (a) at (0,0); \coordinate (c)
       at (2,0); \coordinate (b) at
       (1,1.73); \filldraw (a)
      circle (2pt); \filldraw (b)
      circle (2pt); \filldraw (c)
      circle (2pt); \draw (b) -- (c
      ); \node [below left] at (a)
      {$a$}; \node [above] at (b)
      {$b$}; \node [below right] at
       (c) {$c$}; \end{tikzpicture
      }\vspace{2mm}\end{minipage}}
      \caption{$V=\{a,b,c\},\, E
      =\{(b,c)\}$\hspace{26em}}
144 \end{figure}
```

del sorgente originale da cui ottenere il PDF. Ciò implica che avremo compilato un documento `standalone` contenente il codice compreso tra `\begin{tikzpicture}` e `\end{tikzpicture}` (riga 143) e convertito il relativo PDF nel file `figura1.jpg`, e questo per tutte le figure Ti*k*Z contenute nel documento.

Rimane l'ultimo elemento del progetto didattico: le formule. Con `htlatex` possiamo scegliere se renderle immagini (scelta più sicura per la compatibilità con la maggior parte dei lettori di ebook; peraltro non tutte le formule dovranno subire questo trattamento e il compilatore deciderà quali poter rendere in HTML) o formule MathML (scelta preferibile dal punto di vista della coerenza — la formula è conforme al testo indipendentemente dal ridimensionamento, cosa che non può avvenire con un'immagine — ma insicura dal punto di vista della compatibilità con i lettori di ebook, hardware o software[5]).

4. Le immagini sono rimaste quelle convertite originariamente, rese a una risoluzione uguale a quella dei monitor commerciali.

5. È ovvio che un lettore di ebook hardware integri al proprio interno un software per leggere gli ebook, ma la scelta infelice di linguaggio è volta a distinguere i programmi e le app, installabili su diversi sistemi operativi, dai lettori dedicati.

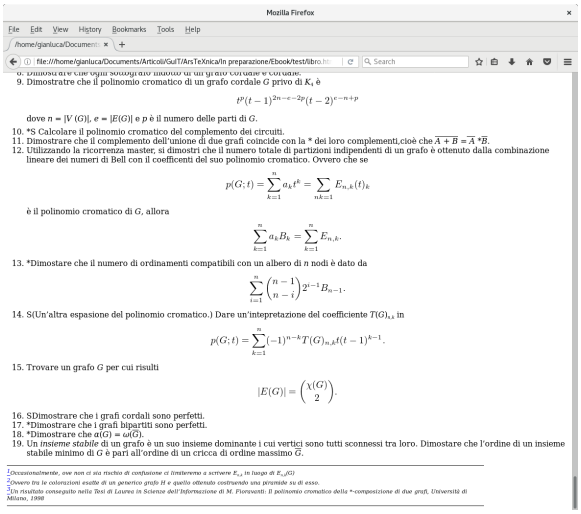


FIGURA 2: Parte finale del documento didattico trasformato in HTML. Le formule sono rese come figure.

Siamo pronti a compilare il sorgente LaTeX per ottenere un HTML. Abbiamo l'accortezza di eliminare l'inclusione di crop commentando la riga 13 del listato 1 e, come visto poc'anzi, di sostituire le figure in Ti*k*Z con le corrispondenti raster. Sia questo file `libro.tex`:

```
htlatex libro "html,fn-in"
```

ci fa ottenere, dopo aver risposto `s` alle eventuali interruzioni per segnalare un errore, un file HTML con tutte le note raggruppate alla fine (senza l'opzione `"html,fn-in"` otterremmo una nota per pagina HTML — in pratica, un file HTML per nota — e, nel caso di testi con molte note, questa non è la soluzione più agevole); parte di questo file è mostrata nella figura 2.[6]

Ci sono comandi che purtroppo non si riescono a tradurre in HTML, forse solo per incompletezza del compilatore. Per esempio, sarebbe facile tradurre `\,` col *thin space* di Unicode (codice U+2009) o anche solo con uno spazio, ma ciò non avviene; per evitare di avere parole attaccate dobbiamo eliminare le occorrenze di `\,` dal sorgente. Anche `\raggedleft` non viene convertito; al suo posto ci conviene usare l'ambiente flushright.

Sono solito aggiustare i titoli, specie nel sommario, usando `\protect\\`. Questo funziona, ma non sono soddisfatto della resa nel sommario dell'HTML, quindi preferisco eliminare le occorrenze di questa sequenza di comandi e lasciare che i titoli vadano a capo quando non hanno più spazio.

6. Avremmo potuto dare il comando `htlatex libro "html,mathml"` col risultato di ottenere un file che il mio Firefox mostra solo fino a un certo punto, pur essendo il sorgente completo, e che, una volta convertito in EPUB, mostra le formule extratestuali in maniera a dir poco fantasiosa.

3 Intervallo: L'HTML è da aggiustare

Quando abbiamo un HTML soddisfacente, cioè che non necessita di essere più rigenerato modificando il sorgente `.tex`, è molto probabile che ci siano piccoli aggiustamenti da fare. I due che faccio riguardano due aspetti. Innanzitutto le spaziature verticali del frontespizio, con l'aggiunta di opportuni tag `<br />`, e la separazione del frontespizio dal retrofrontespizio (inserisco un titolo fantasma: `<h2></h2>`) che altrimenti starebbero sulla stessa pagina. Poi il riposizionamento delle note a piè di pagina. Come abbiamo visto, `htlatex` può essere regolato per mettere le note in fondo al documento HTML, ma il mio datore di lavoro principale le vuole alla fine di ogni capitolo. Per ottenere questo risultato devo tagliare tutte le righe delle note di un capitolo (con l'aiuto di `bc` e di opportune etichette per sapere quante righe tagliare e ritrovare agevolmente il punto in cui tagliare) e copiarle alla fine del relativo capitolo; all'inizio della prima nota di ogni capitolo devo scrivere il tag `<div class="footnotes">` e, alla fine dell'ultima, `</div>`. Fare questo lavoro con le note poste ognuna in un file separato sarebbe stato molto peggio.

Una cosa a cui prestare attenzione è il fatto che i capitoli non numerati non azzerano la numerazione delle note ma la proseguono. Lo stesso avviene con `htlatex`. Naturalmente tutti sanno che basta riazzerare il contatore `footnote` per evitare la numerazione incoerente. Vediamo cosa succede nell'HTML con un MWE (Minimal Working Example):

```
\documentclass[a4paper,11pt]{book}
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage[italian]{babel}
\usepackage{lipsum}
\begin{document}
\chapter*{Prefazione}
\addcontentsline{toc}{chapter}{Prefazione}
\lipsum[1]\footnote{Prima nota.}

\lipsum[2]\footnote{Seconda nota.}
\chapter*{Introduzione}
\addcontentsline{toc}{chapter}{Introduzione}
\setcounter{footnote}{0}
\lipsum[3]\footnote{Prima nota.}

\lipsum[4]\footnote{Seconda nota.}
\chapter{Uno}
\lipsum[5]\footnote{Prima nota.}

\lipsum[6]\footnote{Seconda nota.}
\chapter*{Conclusioni}
\addcontentsline{toc}{chapter}{Conclusioni}
\setcounter{footnote}{0}
\lipsum[7]\footnote{Prima nota.}

\lipsum[8]\footnote{Seconda nota.}
\end{document}
```

Il file HTML risultato della compilazione contiene una serie di link dal testo alle note e viceversa. Vediamo uno per uno i link alle note analizzandone i nomi:

Capitolo	Prima nota	Seconda nota
Prefazione	`fn1x0`	`fn2x0`
Introduzione	`fn1x0`	`fn2x0`
Uno	`fn1x1`	`fn2x1`
Conclusioni	`fn1x1`	`fn2x1`

Da questo capiamo che i richiami a nota di Prefazione e Introduzione puntano entrambi alle stesse note (di Prefazione) e quelli di Uno e Conclusioni puntano alle stesse note (di Uno). Questo succede perché il capitolo numerato incrementa un contatore delle note (la parte `x`⟨*numero*⟩), cosa che non fa un capitolo non numerato. Se non avessimo riazzerato il contatore delle note, non avremmo avuto di questi problemi e avremmo potuto semplicemente correggere a mano i soli numeri dei riferimenti alle note nell'HTML.

A seconda delle lingue impiegate per produrre l'HTML, capita che la codifica non rispecchi il contenuto del file e il browser presenti dei caratteri "fantasiosi". In questo caso è sufficiente cambiare la codifica iniziale dell'HTML (cosa da fare solo in questa fase e mai quando l'HTML è da rigenerare continuamente):

```
<meta http-equiv="Content-Type"
 content="text/html;
 charset=iso-8859-1">
```

presente subito dopo la riga contenente `<head><title>` può presentare un charset inadatto che va verificato ed eventualmente corretto in accordo con la codifica del testo. In alcuni casi ho dovuto usare anche `iconv` per adattare la codifica dell'intero testo.

Codifica giusta o meno, mi sono capitati casi in cui, soprattutto negli indici analitici, le lettere accentate fossero totalmente sbagliate. Uno script si occupa di trattare i casi specifici, per esempio:

```
sed -i 's/\^\^c3\^\^ba/\&uacute;/g'\\
  $file.html
```

corregge tutte le occorrenze di `^^c3^^ba` in ú.

Un'altra cosa di cui tener conto, specie quando il documento contiene del testo in greco politonico, è la mancata "traduzione" di alcuni caratteri e la loro conversione a immagine. Controllate i file generati dalla compilazione con `htlatex` e guardate le immagini PNG. Se ne trovate qualcuna contenente un solo carattere, potete cercare nell'HTML il punto in cui ognuna di esse è caricata e sostituire il relativo codice col codice del carattere in questione (Unicode Consortium, 2018, è fondamentale avere a portata di mano le tabelle dei caratteri Unicode).

Nel caso in esame, cioè il progetto didattico, l'HTML non ha bisogno di ulteriori aggiustamenti.

4 Secondo tempo: da HTML a EPUB (o MOBI)

Una volta che siamo soddisfatti dell'HTML, possiamo generare l'ebook. Di questo si occuperanno Calibre e i suoi comodissimi filtri da riga di comando (Goyal, 2018):

`ebook-convert` ⟨*file.html*⟩ ⟨*file.epub*⟩ `--cover` ⟨*jpeg della copertina*⟩ `--no-default-epub-cover --title` ⟨*titolo*⟩ `--authors` ⟨*autore*⟩ `--publisher` ⟨*editore*⟩

se vogliamo un file `.epub`,

`ebook-convert` ⟨*file.html*⟩ ⟨*file.mobi*⟩ `--cover` ⟨*jpeg della copertina*⟩ `--no-inline-toc --title` ⟨*titolo*⟩ `--authors` ⟨*autore*⟩ `--publisher` ⟨*editore*⟩

se vogliamo un file `.mobi`.[7]

Per generare l'EPUB diamo il seguente comando (tutto di seguito, anche se qui va a capo per motivi di spazio) dal terminale:

```
ebook-convert libro.html libro.epub
--cover copertina_dantona.jpg
--no-default-epub-cover
--title "Introduzione alla matematica
discreta"
--authors "Ottavio D'Antona"
--publisher Apogeo
```

Ecco come Calibre mostra il libro generato nella biblioteca interna (figura 3) e come leggiamo l'ebook (figura 4).

Durante il processo di conversione da HTML a EPUB Calibre ci informa che, dopo "`Splitting markup on page breaks and flow limits, if any...`" e "`Looking for large trees in libro.html...`", "`No large trees found`", cioè in questo documento non ci sono capitoli talmente lunghi da dover essere ulteriormente suddivisi.

7. Il servizio Kindle Direct Publishing di Amazon accetta senza remore un EPUB (senza copertina) e un file di copertina per generare da sé un file `.mobi`. Non ho mai approfondito la questione della validazione del formato MOBI, che sembra essere una faccenda del tutto proprietaria. In ogni caso, tenete presente che tale formato non supporta MathML per definizione.

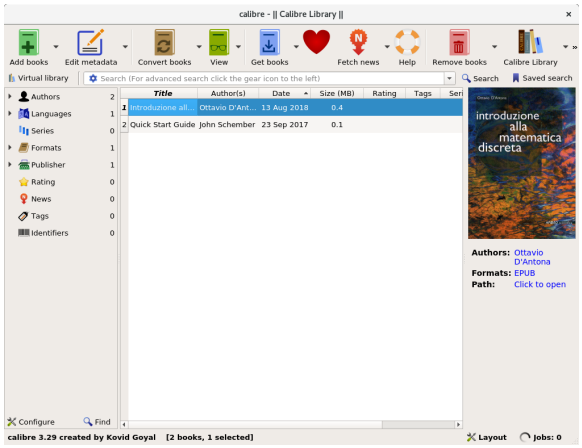


Figura 3: L'ebook è stato importato in Calibre, pronto per essere letto.

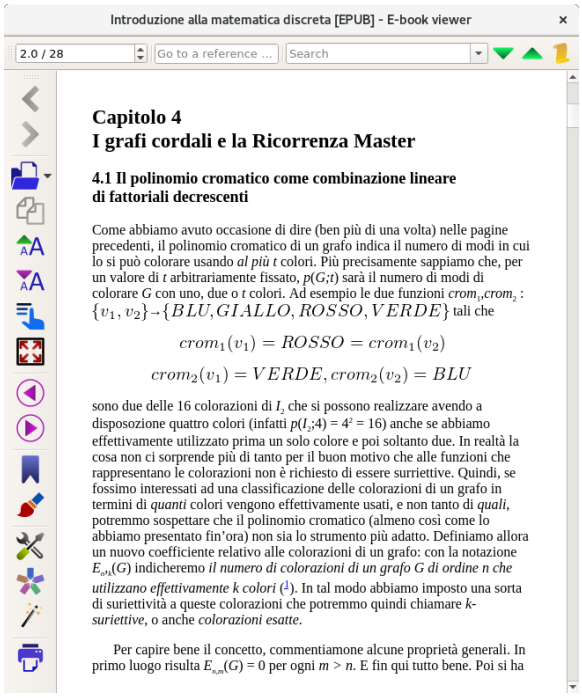


Figura 4: Il lettore di Calibre mostra l'ebook al suo meglio.

Può capitare che gli *split*[8] in cui viene suddiviso il file HTML per ottenere l'EPUB non possano contenere un intero capitolo, e quindi ci sia più di uno split per capitolo. Questo provoca il fastidioso effetto di vedere un titolo di livello inferiore al capitolo cominciare in una pagina nuova, magari dopo un bel bianco. Qui viene in aiuto Sigil, che permette di fare il merge degli split, ma con potenziali problemi di cui parleremo nella sezione 5.

Recentemente mi è capitato un caso che a tutta prima mi sembrava un bug di Calibre: la generazione dell'EPUB si interrompeva con errori a causa di un *large tree* (un capitolo lungo) che Calibre non riusciva a suddividere. Dopo aver inutilmente

8. Ogni *split* contiene un elemento che inizia con un titolo, reale o fantasma, di qualunque livello. La suddivisione di default è uno split per capitolo.

aggiornato Calibre mi sono reso conto del motivo vero: il capitolo conteneva una lunghissima bibliografia, in sostanza una lunga lista, che non poteva essere suddivisa per mancanza di titoli. L'elenco infatti non prevedeva sezioni tematiche ma solo una lista alfabetica, lunga ed estenuante.

La soluzione è consistita nell'interrompere manualmente (editando l'HTML) la lista raggruppando tutti gli autori con la stessa iniziale in una lista a sé e, tra la fine di una lista e l'inizio della successiva, mettere un titolo fantasma (nel caso specifico, `<h3></h3>`). Questo ha permesso di ottenere diversi split che poi è bastato riunire col merge di Sigil.

5 Attualità: validazione dell'EPUB

L'ultima fase della produzione di un ebook consiste nella sua validazione, cioè nel controllo che tutti gli elementi del markup siano conformi allo standard. In tale fase è fondamentale `epubcheck`. Questo è un file Java che, ricevuto in input il file EPUB da validare, restituisce in output una lista di errori (con relativi numeri di riga e colonna) o la scritta "Nessun errore riscontrato". Nel caso in cui ci siano errori da correggere (evento tutt'altro che raro), trovo utilissimo usare Sigil come editor in luogo del mio prediletto `vi`.

Torniamo al progetto didattico: Calibre non ha fatto una piega nell'aprire e mostrarci l'EPUB ottenuto, ma siamo sicuri che potremmo metterlo in vendita senza problemi? Per saperlo dobbiamo validarlo ricorrendo a `epubcheck`:

```
java -jar /opt/epubcheck-4.0.2/
epubcheck.jar libro.epub
```

(sul mio sistema il `.jar` si trova sotto `/opt/epubcheck-4.0.2/`). Il responso è un insieme di sei errori, tutti del tipo "`element "table" not allowed here; expected the element end-tag, text or element "a", "abbr", "acronym", "applet", "b", "bdo", "big", "br", "cite", "code", "del", "dfn", "em", "i", "iframe", "img", "ins", "kbd", "map", "noscript", "ns:svg", "object", "q", "samp", "script", "small", "span", "strong", "sub", "sup", "tt" or "var" (with xmlns:ns="http://www.w3.org/2000/svg")`" in diversi punti di `libro.epub`.

Per tentare di correggerli apriamo l'ebook con Sigil. La prima cosa che Sigil segnala tramite una finestra di dialogo è il fatto che l'EPUB ha il relativo HTML mal formato e può provare a correggere i file automaticamente, sebbene ciò potrebbe causare una perdita di dati. Proviamo a rispondere affermativamente alla richiesta di correzione automatica: se va bene, siamo fortunati; se va male, possiamo sempre ripartire dalla conversione HTML → EPUB e fare il resto manualmente.

In effetti, Sigil si apre mostrando i due split (copertina e unico capitolo) il cui contenuto sembra tutto a posto. (Per scrupolo ho provato ad aprire l'ebook senza farlo correggere e tutte le formule incluse come immagini e le immagini dei grafi non venivano visualizzate da Sigil; Calibre non aveva problemi di sorta. La tabella aveva un aspetto diverso da quello atteso.) Se salviamo e riproviamo la validazione non otteniamo più errori. In questo caso siamo stati molto fortunati. Nel 100% delle situazioni lavorative non sono stato altrettanto fortunato. Infatti, la procedura a cui di solito sottopongo gli ebook in lavorazione è: 1) apertura dell'EPUB con Sigil (che in genere non riscontra errori); 2) controllare ogni messaggio di errore di `epubcheck` e i relativi numeri di split, di riga e di colonna;[9] 3) aprire lo split e visualizzarne il codice HTML; 4) tentare di correggere l'errore modificando il codice, magari facendo ricorso ai manuali dello standard per capire la natura dell'errore o la sua soluzione[10]. Se la successiva validazione (ricordiamoci di salvare il file modificato) non dà più l'errore, la correzione è riuscita.

L'errore che in assoluto mi capita più spesso (e che mi fa pensare al perché negli ambienti commerciali Calibre è visto in malo modo come strumento inaffidabile) è il posizionamento di un riferimento ipertestuale in un punto non consentito. Nella figura 5 ho evidenziato la linea 17 del sorgente di un ebook di recente realizzazione. Quella linea è corretta ma il file originario conteneva

```
<h2 class="calibre2" id="calibre_pb_3">
Indice</h2>
<a id="x1-1000" class="calibre3"></a>
```

e il validatore diceva che un tag `<a>` in quel punto non è ammesso. Da lì lo spostamento nel punto mostrato nella figura 5.

Sempre nella figura 5, sulla sinistra, vediamo l'elenco degli split in cui è suddiviso il libro. Dopo lo split 7 c'è il 12. Ciò vuol dire che abbiamo attaccato insieme (merge) gli split 7–11. Questo causa un errore in `toc.ncx` perché i riferimenti agli split ivi contenuti non vengono corretti. Dunque `toc.ncx` continua a contenere i riferimenti agli split 8–11 che ora non esistono più. Dovremo aprire il codice del sommario e correggere a mano i riferimenti non più esistenti, scrivendo `split_007` dove compare `split_008`–`split_011`. Naturalmente il validatore ci assiste anche per questo tipo di errore.

9. A volte succede che le coordinate di riga e colonna del primo errore siano sbagliate — lo split non lo è mai — e che, di conseguenza, lo siano tutte le altre. Corretto il primo errore, la nuova validazione darà tutte le coordinate corrette.

10. Ci può venire in soccorso il sito International Digital Publishing Forum (2018), unitamente a W3C (2018).

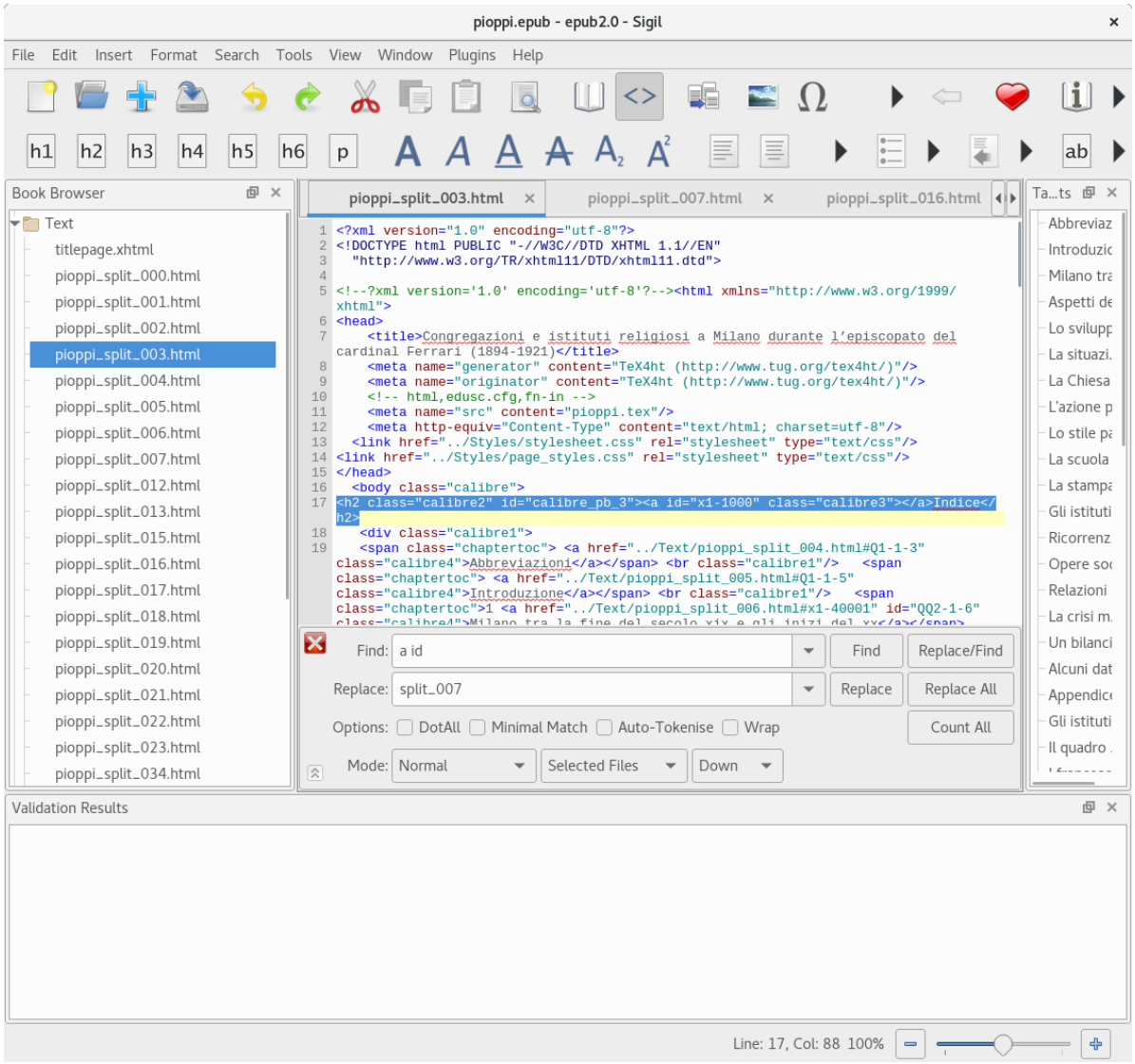


FIGURA 5: Elenco degli split e codice di uno di essi così come mostrati da Sigil.

6 Un editor di ebook alternativo

Finora ho parlato di Sigil perché è veramente lo strumento che uso. Da qualche giorno sto valutando la sua sostituzione con l'editor di Calibre (`ebook-edit`). Mi pare opportuno elencare brevemente alcune ragioni per usare l'uno o l'altro.

Sigil permette di editare l'ebook sia modificando l'HTML che modificando direttamente la versione visuale. Però consente di visualizzare alternativamente nell'editor l'ebook o il suo codice sorgente, visualizzazione che cambiamo agendo sullo switch dedicato. Quando apriamo un file EPUB viene comunque mostrato il contenuto del primo split e le barre dei pulsanti – quella delle operazioni sul file e quella di editing – sono sempre presenti. Non è possibile chiudere tutti gli split: uno di essi rimane comunque aperto nell'editor. Il navigatore, a sinistra, mostra tutti gli split, gli stili, le immagini e tutto quanto componga l'ebook. Come già detto, il validatore adottato è il sorpassato FlightCrew, in genere molto generoso nel perdonare elementi attualmente fuori standard.

L'editor di Calibre, `ebook-edit`, permette la sola modifica dell'HTML ma, siccome visualizza il codice sorgente nell'editor parallelamente a un'anteprima dell'ebook, le modifiche si propagano dall'editor all'anteprima in tempo quasi reale.[11] L'unica interazione che possiamo avere nella finestra di anteprima è evidenziare un brano di testo per fare in modo che il cursore si posizioni all'inizio dei relativi tag nell'editor del codice. Quando apriamo per la prima volta un file EPUB l'editor è vuoto e solo la finestra del navigatore ci mostra il contenuto dell'ebook: split, stili, immagini ecc. La barra dei pulsanti mostra solo quelli relativi alle operazioni generali sul file. La barra dei pulsanti di editing

11. Ricordiamo che il concetto di *tempo reale* è un concetto mutuato dai sistemi operativi che indica la certezza di avere una determinata operazione completata entro un tempo limite non dilazionabile per alcuna ragione. Non ha niente a che vedere con la volgarizzazione dell'espressione che vorrebbe significare la contemporaneità.

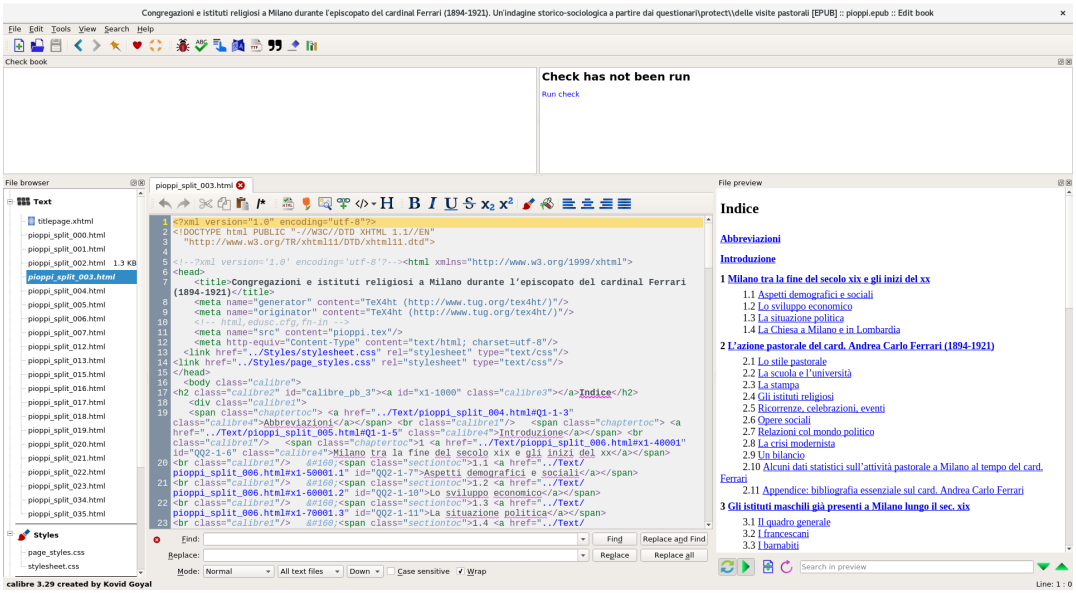


FIGURA 6: L'editor di ebook di Calibre.

è visibile solo quando apriamo uno split per l'editing ed è posta nella stessa scheda del codice dello split. Anche `ebook-edit` ha un validatore non specificato nei manuali ma che, genericamente, è dichiarato conforme all'HTML 5. Nella figura 6 vediamo l'interfaccia di `ebook-edit` all'opera sullo stesso documento della figura 5.

Personalmente, non usando l'editor visuale di Sigil (che trovo alquanto incomprensibile), sto valutandone l'abbandono in favore di `ebook-edit`. Altri utenti potrebbero addurre ragioni contrarie e fare la scelta opposta alla mia.

7 Conclusioni

Produrre un ebook in formato EPUB a partire da un file LATEX non è un'operazione immediata né economica: serve tempo per ognuna delle fasi di lavorazione, serve aver accumulato esperienza per risolvere agevolmente gli intoppi sempre possibili, e serve avere almeno una cosiddetta *working knowledge* degli strumenti e dei linguaggi coinvolti per non aver voglia di cambiare mestiere.

A queste oggettive difficoltà fa da contraltare il controllo assoluto sul prodotto finale: possiamo cambiare l'HTML, il CSS e l'EPUB fino a ottenere il prodotto finale che meglio corrisponde alle nostre aspettative e alle esigenze del committente. L'ulteriore vantaggio è che l'EPUB così generato tende a essere più piccolo degli EPUB generati — pur in meno tempo — con i più noti prodotti commerciali.

Riferimenti bibliografici

D'ANTONA, O. (1998). *Introduzione alla matematica discreta*. Apogeo scientifica. Apogeo, Milano.

EPUBCHECK TEAM (2018). «GitHub - IDPF/epubcheck: Validation tool for EPUB». URL https://github.com/idpf/epubcheck.

GOYAL, K. (2018). «Command line interface — calibre 3.30.0 documentation». URL https://manual.calibre-ebook.com/generated/en/cli-index.html.

GURARI, E. M. (2004). «TEX4ht: HTML Production». *TUGboat*, **25** (1). Proceedings of the Practical TEX 2004 Conference.

HOFTICH, M. (2016). «Github - michal-h21/lua4ht: experimental package for direct comversion from latex to html using tex4ht and lualatex». URL https://github.com/michal-h21/lua4ht.

INTERNATIONAL DIGITAL PUBLISHING FORUM (2018). «The Trade and Standards Organization for the Digital Publishing Industry». URL http://idpf.org/.

PIGNALBERI, G. (2015). «LATEX in un'editrice universitaria: come e perché book non è adatta e una possibile alternativa funzionante». *ArsTEXnica*, (19), pp. 33–41.

SCHEMBER, J. (2018). «Sigil Ebook». URL https://sigil-ebook.com/.

UNICODE CONSORTIUM (2018). «Unicode 11.0 Character Code Charts». URL https://www.unicode.org/charts/.

W3C (2018). «HTML5 Reference». URL https://dev.w3.org/html5/html-author/.

▷ Gianluca Pignalberi
g dot pignalberi at gmail dot com

Introduzione al pacchetto xparse

Claudio Beccari

Sommario

Il pacchetto xparse mette a disposizione un certo numero di macro avanzate per definire nuovi comandi e nuovi ambienti con una grande varietà di argomenti. Queste funzionalità sono preziose sia per chi scrive file di estensione sia per gli utenti finali.

Abstract

The xparse package allows to use a number of advanced macros to define new commands and environments with a large variety of arguments. Such facilities are very useful for both package writers and end users.

1 Introduzione

Ogni utente LaTeX sa che il linguaggio permette di definire nuovi comandi e nuovi ambienti usando i comandi di alto livello seguenti:

```
\newcommand
\renewcommand
\providecommand
\newenvironment
\renewenvironment
```

Il comando `\provideenvironment` manca dal nucleo di LaTeX. I comandi `\newcommand` e `\newenvironment` definiscono rispettivamente un nuovo comando o un nuovo ambiente che non sia stato già definito, altrimenti emettono un messaggio d'errore e non eseguono la definizione specificata nel loro argomento. I comandi `\renewcommand` e `\renewenvironment` ridefiniscono rispettivamente un comando o un ambiente che sia già stato definito; in caso contrario, segnalano l'errore e non procedono alla ridefinizione specificata. `\providecommand` definisce un nuovo comando che non sia stato già definito, ma non fa nulla se il comando esiste già.

Per definire comandi che accettino anche un asterisco finale bisogna ricorrere sostanzialmente a lunghi giri di macro che controllino se il comando è seguito da un asterisco e, a seconda dell'esito del controllo, attivano altri comandi interni a cui eventualmente forniscono argomenti predefiniti.

Questa interfaccia fra i comandi di definizione di alto livello e i comandi nativi che effettivamente eseguono quelle definizioni o ridefinizioni sono molto utili per il programmatore e, ancora di più, per l'utente finale che voglia definirsi comandi propri. Ma i comandi così definiti sono un po' rigidi e non permettono di fare più di tanto. Infatti:

- non consentono di definire comandi con argomenti delimitati;
- non consentono di definire comandi con più di un argomento facoltativo; questo inoltre può essere racchiuso solo fra parentesi quadre;
- non consentono di definire comandi sempre robusti;
- lo stesso vale per i comandi di apertura degli ambienti, con l'ulteriore limitazione che gli argomenti all'apertura dell'ambiente sono disponibili, appunto, solo per i comandi di apertura.

Per superare questi limiti, il programmatore deve adoperare i comandi di basso livello nativi del sistema TeX e deve saper maneggiare correttamente i comandi locali o globali e quelli che sviluppano o meno gli eventuali argomenti costituiti da macro. Non impossibile, ma è richiesta una certa esperienza per fare queste cose e tanta pazienza per correggere gli eventuali errori. Per definire comandi robusti l'utente deve ricorrere all'ulteriore comando `\DeclareRobustCommand` che però non esegue nessuna verifica sulla eventuale preesistenza del comando che si vorrebbe dichiarare robusto.

Da alcuni anni il LaTeX 3 Team sta creando un linguaggio intermedio fra quello nativo del sistema TeX e il mark up di alto livello di LaTeX. Questo linguaggio prende la sigla "L3"; e nei passati numeri di questa rivista Enrico Gregorio ha esposto diverse funzionalità di questo linguaggio.

La potenza di L3 è enorme ma di contro la sua sintassi è complessa e delicata. Nonostante le difficoltà d'uso, il linguaggio semplifica l'opera dei programmatori che non hanno più bisogno di inventarsi ogni volta come affrontare i problemi di programmazione che possono sorgere per costruzioni tipografiche complesse. Già da tempo molti pacchetti di uso generale sono stati tradotti in L3, facilitando la loro manutenzione e consentendo ulteriori funzionalità.

Il pacchetto xparse si pone appunto come interfaccia tra l'utente finale o il programmatore e il linguaggio L3, in modo da eliminare ogni difficoltà nella definizione di nuovi comandi e nuovi ambienti con la massima libertà, togliendo le limitazioni dei comandi nativi di LaTeX. Infatti, i nuovi comandi di definizione:

- consentono di usare argomenti delimitati fra delimitatori che l'utente stabilisce a suo piacimento;

- consentono di definire comandi con diversi argomenti facoltativi diversamente delimitati e a cui eventualmente è assegnato un valore preimpostato;
- consentono di definire comandi che sono sempre robusti;
- per gli ambienti gli argomenti obbligatori e facoltativi dello statement di apertura sono disponibili anche durante l'esecuzione dei comandi di chiusura;
- consentono di definire a piacere dell'utente argomenti booleani in modo simile a quanto fa l'asterisco per i comandi nativi di LaTeX.

.

In sostanza, usando le funzionalità del pacchetto xparse la definizione di una sola macro consente di fare in un colpo solo quello che con i comandi nativi o i comandi del nucleo di LaTeX richiede la definizione di catene di macro e di test, che spesso si rivelano fragili e danno luogo a errori difficili da rilevare e correggere.

2 Le macro del pacchetto di xparse

Le macro di xparse per definire nuovi comandi o ambienti ricorrono a un concetto nuovo: la lista dei *descrittori degli argomenti*. Invece di specificare solamente il numero degli argomenti, come avviene con i comandi del nucleo di LaTeX, bisogna fornire obbligatoriamente una lista di codici che descrivono ciascun argomento, ne specificano i delimitatori, se sono obbligatori o facoltativi, un eventuale valore predefinito, se sono di tipo booleano o se accettano un valore. Vedremo fra poco questa lista. Secondo la terminologia del linguaggio L3, comandi e ambienti si chiamano "funzioni"; le funzioni di tipo "command" agiscono sui loro argomenti; quelle di tipo "environment" agiscono sui loro argomenti e su quanto contenuto fra `\begin` e `\end`.

Ora vediamo invece la serie di comandi a disposizione.

`\NewDocumentCommand` definisce un nuovo comando.

`\RenewDocumentCommand` ridefinisce un preesistente comando.

`\ProvideDocumentCommand` provvede alla definizione di un comando se non è mai stato definito prima.

`\DeclareDocumentCommand` definisce un comando indipendentemente dal fatto che sia nuovo o preesistente.

`\NewDocumentEnvironment` definisce un nuovo ambiente.

`\RenewDocumentEnvironment` ridefinisce un ambiente preesistente.

`\ProvideDocumentEnvironment` provvede a definire un ambiente se non è mai stato definito prima.

`\NewExpandableDocumentCommand` definisce un nuovo comando espandibile.

`\RenewExpandableDocumentCommand` ridefinisce un comando preesistente espandibile.

`\ProvideExpandableDocumentCommand` se non era mai stato definito prima, provvede a definire un comando espandibile.

`\DeclareExpandableDocumentCommand` indipendentemente dal fatto che sia nuovo o preesistente definisce un comando espandibile.

Tutti i comandi che vengono definiti sono robusti; solo con gli ultimi quattro comandi di definizione essi sono sviluppabili; dovrebbero essere usati con la massima cautela e solo se veramente necessario; *se non si sa se l'espansione sia assolutamente necessaria, non si usino queste definizioni.* In sostanza questi ultimi quattro comandi sono per programmatori esperti, non per utenti finali che potrebbero non avere chiara la necessità di espandere certi comandi.

Oltre a questi comandi, ce ne sono alcuni altri che sono di uso decisamente frequente, mentre altri sono più specializzati e vanno studiati nella documentazione di xparse, (The LaTeX Project Team, 2018). I comandi frequenti sono i seguenti.

`\IfNoValue` verifica se un argomento facoltativo non ha ricevuto alcun valore.

`\IfValue` verifica se un argomento facoltativo ha ricevuto un valore.

`\IfBoolean` verifica se un argomento booleano è presente.

Tutti e tre questi comandi hanno una sintassi del tipo:

```
\⟨comando⟩TF{⟨argomento⟩}{⟨vero⟩}{⟨falso⟩}
```

Il suffisso `TF` può ridursi anche solo a `T` per indicare "true" o a `F` per indicare "false". Chi scrive preferisce sempre usare entrambe le uscite del test e se fosse necessario lascerebbe vuoto il campo ⟨*true*⟩ o quello ⟨*false*⟩. I primi due comandi servono per verificare se un dato ⟨*argomento*⟩, indicato con il suo numero d'ordine nella forma `#`⟨*numero*⟩, ha o non ha ricevuto un valore e a seconda del caso esegue il ramo ⟨*vero*⟩ oppure ⟨*falso*⟩. Il terzo comando serve per verificare se un argomento booleano è presente o assente; il risultato del test è vero solo se l'argomento booleano è presente. Ovviamente questi comandi sono da usare nel corpo della definizione delle macro indicate dai comandi specifici per definirli.

La sintassi dei comandi di definizione è la seguente

```
\⟨prefissi⟩command{⟨descrittori⟩}{⟨definizione⟩}
\⟨prefissi⟩environment{⟨descrittori⟩}{⟨apertura⟩}%
    {⟨chiusura⟩}
```

Va notato che i ⟨*descrittori*⟩ sono generalmente una lista di codici separati da spazi (non da virgole) e identificano nell'ordine fino a nove argomenti (il massimo accessibile dai comandi nativi del sistema TEX sottostante); per le definizioni degli ambienti essi sono accessibili con lo stesso "numero" sia nei comandi di ⟨*apertura*⟩ sia nei comandi di ⟨*chiusura*⟩.

Va ancora notato che gli argomenti booleani (come per esempio l'asterisco) sono dei veri e propri argomenti e dunque contano nel loro novero. Per i comandi la cosa non è tanto diversa da come avviene per l'asterisco dei comandi di LATEX; invece per gli ambienti gli argomenti booleani, e quindi gli asterischi, sono veri argomenti e non fanno parte del nome dell'ambiente; questo è un po' diverso da quanto accade con i comandi del nucleo di LATEX, dove l'asterisco eventuale fa parte del nome dell'ambiente. Tanto per essere più chiari, gli ambienti figure e figure* sono due ambienti distinti nelle definizioni del cuore di LATEX, quindi, se è stato iniziato l'ambiente asteriscato con `\begin{figure*}`, esso deve deve venire chiuso con `\end{figure*}`. Con gli ambienti di xparse, l'eventuale asterisco va messo dopo l'apertura dell'ambiente; se figure fosse definito con i comandi di xparse la sua apertura si farebbe con `\begin{figure}*` e la chiusura con `\end{figure}`. Vedi più avanti anche l'esempio 4.3.

Questa particolarità potrebbe essere d'impiccio per molti utenti finali, meno per i programmatori, per l'abitudine che hanno i primi a usare la stessa stringa nei comandi di apertura e in quelli di chiusura.

3 La lista degli argomenti

La lista degli argomenti è semplicemente una lista di codici, facoltativamente separati da spazi, che descrivono le particolarità di ogni argomento nello stesso ordine in cui vengono inseriti dall'utente fra gli argomenti delle macro che egli usa.

Come si evince dai paragrafi precedenti, i descrittori sono numerosi.

Argomenti obbligatori

`m` indica un argomento *mandatory*, obbligatorio; dello stesso tipo dell'argomento di una normale macro LATEX.

`r`⟨*car1*⟩⟨*car2*⟩ indica un argomento *required*, obbligatorio, delimitato a sinistra dal carattere ⟨*car1*⟩ e a destra dal carattere ⟨*car2*⟩; nel cuore di LATEX questo genere di argomenti è piuttosto raro; si può ricordare l'argomento contenuto fra parentesi tonde che indica la coppia di coordinate dei punti nelle macro che si usano negli ambienti di disegno programmato; per esempio negli ambienti picture, tikzpicture e simili. Ma il nucleo di LATEX non dispone di comandi per definire argomenti delimitati che, quindi, sono definiti ricorrendo ai comandi nativi di TEX.

`v` indica un argomento da leggere "verbatim", racchiuso fra il primo e l'ultimo carattere di una stringa, allo stesso modo dell'argomento del comando `\verb`. Questo carattere delimitatore può essere un carattere qualsiasi, purché non faccia parte della stringa da leggere verbatim, né sia uno dei seguenti caratteri speciali: `\`, `#`, `{`, `}`, o ␣. Questo descrittore può servire per definire una macro che contiene una stringa verbatim, ma ha un certo numero di limitazioni e deve essere considerato sperimentale, secondo quanto affermato nella stessa documentazione di xparse. Qui non ne parlerò più, anche perché il suo uso mi sembra troppo limitato.

Argomenti facoltativi

`o` (lettera `o` minuscola) indica un normale argomento da introdurre fra parentesi quadre senza un valore predefinito; per sapere se l'argomento è stato usato e gli è stato assegnato un valore, bisogna controllarlo con la funzione `\IfValue` oppure `\ifNoValue` (con il suffisso `TF`).

`d`⟨*car1*⟩⟨*car2*⟩ indica un argomento facoltativo delimitato a sinistra dal carattere ⟨*car1*⟩ e a destra da ⟨*car2*⟩; non gli viene assegnato nessuna valore di default. Anche in questo caso se ne verifica il valore mediante la funzione `\IfValue` oppure `\ifNoValue`.

`O{`⟨*valore*⟩`}` (lettera `O` maiuscola) indica un argomento facoltativo delimitato da quadre a cui è assegnato ⟨*valore*⟩ come impostazione predefinita.

`D`⟨*car1*⟩⟨*car2*⟩`{`⟨*valore*⟩`}` indica un argomento delimitato come nel caso `d` e con un ⟨*valore*⟩ predefinito come nel caso `O`.

Argomenti booleani

`s` indica un argomento booleano costituito da una "stella", cioè un asterisco. Se ne verifica la presenza mediante la funzione `\IfBoolean` con il debito suffisso `TF`.

`t`⟨*car*⟩ Indica un "token" facoltativo costituito dal carattere ⟨*car*⟩. Funziona come l'asterisco, ma è costituito da un carattere specifico scelto dal programmatore.

Argomenti facoltativi "deprecati" Esistono alcuni descrittori di argomenti facoltativi che la documentazione di xparse raccomanda di non usare più, ma che sono mantenuti per compatibilità con il passato. Chi scrive ne ha usati alcuni in alcune sue macro e non ha mai riscontrato

inconvenienti; elenca qui di seguito solo quelli che effettivamente ha usato.

`g` descrive un argomento facoltativo delimitato da graffe e senza un valore predefinito. Si capisce subito perché delimitare con graffe alcuni argomenti facoltativi possa ingenerare confusione; tuttavia con la debita attenzione lo si può fare.

`G{`⟨*valore*⟩`}` descrive un argomento facoltativo delimitato da graffe con il ⟨*valore*⟩ predefinito; continua a valere la perplessità circa l'uso delle graffe per delimitare argomenti facoltativi; tuttavia, essendo possibile assegnare loro un ⟨*valore*⟩ predefinito, i rischi si riducono in modo considerevole. Nel seguito non presenterò esempi che usino questi descrittori, rispettando la raccomandazione del LaTeX Project Team di non farne uso.

Esistono altri ⟨*descrittori*⟩ di uso raro e comunque di carattere sperimentale.

Come si vede, dunque le funzioni definite da xparse accettano più di nove tipi diversi di argomenti: obbligatori o facoltativi; regolari o delimitati in modo specifico, con o senza valori predefiniti; con contenuto variabile o con valore booleano.

Dietro le quinte il linguaggio L3 trasforma queste funzioni in comandi nativi del sistema TeX; per gestire questa moltitudine di forme di definizione con il comandi normali del nucleo di LaTeX un programmatore si romperebbe la testa. Il LaTeX 3 Team si è dato da fare in modo incredibile; se l'utente si prendesse la briga di tracciare il lavorio che il programma di compilazione esegue dietro le quinte, troverebbe senz'altro molte più operazioni di quelle che un programmatore normale potrebbe cercare di fare usando il proprio intelletto. D'altra parte oggi i calcolatori sono così veloci che risparmiare sul numero delle operazioni da compiere per trasformare il codice sorgente in linguaggio macchina è diventato abbastanza superfluo: la velocità di compilazione di un dato documento dipende più dagli accessi ai dischi meccanici che non dalla CPU della macchina.

4 Esempi

Quando si parla di software molto potente è difficile presentare esempi significativi e non troppo complessi ma che descrivano bene le funzionalità che si desiderano illustrare.

Tuttavia un software come il pacchetto xparse potrebbe risultare difficile anche per un programmatore esperto. Questo, in qualità di esperto, sarebbe in grado di sperimentare autonomamente e in poco tempo potrebbe accumulare sufficienti conoscenze da potersi definire "esperto" anche di xparse. Un utente finale potrebbe invece essere intimorito da un linguaggio molto tecnico e potrebbe avere delle difficoltà a cimentarsi con il software in esame.

Presenterò quindi degli esempi reali, che discendono dalla mia esperienza diretta.

4.1 La scacchiera/cruciverba

Un esempio di applicazione delle funzionalità di xparse appare anche nell'articolo (Beccari *et al.*, 2018) dove viene definita con le funzionalità di xparse una funzione che accetta, oltre all'asterisco facoltativo, altri quattro argomenti diversamente delimitati di cui uno solo obbligatorio; alcuni di questi argomenti in realtà sono liste di valori e non tutti sono obbligatori. Facendo correttamente il conteggio delle possibilità che offrono gli argomenti facoltativi e la presenza o assenza di alcuni valori dalle liste, esistono ben sette elementi facoltativi che possono essere specificati oppure omessi; non tutti sono sensati, nel senso che hanno una certa ridondanza, ma in teoria esistono 128 modi diversi di specificare gli argomenti. Nessuna macro del nucleo di LaTeX consente una tale flessibilità.

Non riporto qui il codice del comando `\scacchiera` perché pubblicato nel numero precedente di questa rivista: questa sì che sarebbe una grossa ridondanza.

4.2 I comandi per l'indice analitico di una classe per guide su LaTeX

La classe GuidaLC serve per comporre certe brevi guide tematiche, non ancora pubblicate dal GUIT; evidentemente questa classe deve avere molte funzionalità specifiche adatte all'argomento di queste guide, che servono per descrivere un linguaggio formale, il mark up di LaTeX. Bisogna scrivere nel testo i nomi di molte macro, ma non bisogna svilupparle; si descrivono gli ambienti, i pacchetti e tante altre cose tipicamente legate a LaTeX. I nomi di questi oggetti vengono tutti riportati nell'indice analitico; sarebbe tuttavia oneroso scrivere ogni volta il comando per il testo e quello per l'indicizzazione duplicando sostanzialmente le stesse informazioni.

Prendiamo l'esempio delle macro, o *control sequence*, per scrivere le quali basta definire il comando `\cs`; spesso è necessario inviarne il nome anche all'indice analitico e altre volte basta solo scriverne il nome. Ecco allora che un comando asteriscato potrebbe essere utile: senza asterisco il comando scrive il nome e lo invia all'indice, mentre con l'asterisco scrive solo il nome.

Questo è un problema facile da risolvere; infatti basta definire una semplice funzione con xparse. Per scrivere una macro in modo che non venga sviluppata è necessario privarla della barra iniziale in modo da usarne solo il nome: è compito della macro da definire quello di prefissare il nome con la barra inversa; nello stesso tempo se quella macro deve andare a finire nell'indice analitico deve essere indicizzata col solo nome senza la barra inversa.

Pertanto per scrivere e indicizzare la control sequence `\LaTeX`, per esempio, bisogna usare il comando `\cs{LaTeX}` mentre per scriverne soltanto il nome senza indicizzarlo bisogna usare il comando asteriscato `\cs*{LaTeX}`.

Separiamo le due cose: una macro che specifica lo stile di scrittura e l'altra che agisce come l'eventuale presenza dell'asterisco richiede.

Ecco il codice:

```
\DeclareRobustCommand*\csstyle[1]{{%
   \normalfont\texttt{\char92#1}%
}}

\NewDocumentCommand{\cs}{s m}{%
   \csstyle{#2}\IfBooleanTF{#1}{}{%
   \index{#2@\csstyle{#2}}}}
```

Come si vede, la lista di descrittori si riduce solo a due elementi, il primo, `s`, per l'asterisco facoltativo; il secondo, `m`, per ricevere solo il nome della macro da scrivere ed eventualmente indicizzare.

4.3 Un ambiente asteriscato

È noto che il nucleo di LaTeX mette a disposizione dell'utente un certo numero di ambienti asteriscati; quello forse più usato è l'ambiente figure* per comporre figure a giustezza piena in documenti composti a due colonne. Va rilevato che l'asterisco è parte del nome dell'ambiente; quindi esso, l'asterisco, va ripetuto nel comando di chiusura.

Con le funzionalità di xparse è possibile definire ambienti in cui l'asterisco è un parametro booleano facoltativo e non fa parte del nome dell'ambiente. Un esempio è l'ambiente ThesisTitlePage del pacchetto TOPtesi, (Beccari, 2018b). La sua definizione è la seguente, dove per semplicità sono eliminati la maggior parte dei comandi di apertura e di chiusura, che si possono eventualmente esaminare nel dettaglio nella documentazione in inglese di quel pacchetto, (Beccari, 2018b).

```
\NewDocumentEnvironment{ThesisTitlePage}{s}
{% APERTURA
\IfBooleanTF{#1}{\boolfalse{topTPTlogos}}%
                {\booltrue{topTPTlogos}}%
   \begin{titlepage}
   ....
}{% CHIUSURA
   \ifbool{topTPTlogos}{...}{...}
   ...
   \end{titlepage}
   \newpage
% Legal/Copyright page
   \ifdefempty{...
   ...
   \newpage}
}
```

Come si vede la lista degli argomenti prevede solo un asterisco facoltativo; a seconda della presenza dell'asterisco, il logo dell'ateneo viene messo in testa oppure al centro della pagina del titolo; finita la composizione della pagina del titolo, viene eventualmente composta la pagina legale a seconda che una certa macro (che potrebbe contenere la dichiarazione di copyright) sia una stringa vera e propria o sia vuota. I comandi `\boolfalse`, `\booltrue` e `\ifdefempty` sono macro disponibili con il pacchetto etoolbox e il cui significato è trasparente. Invece la funzione `\IfBooleanTF`, di xparse, permette di controllare la presenza dell'asterisco facoltativo per impostare a vero o falso la variabile booleana `topTPTlogos`. Tutto come previsto. Ma la particolarità è che l'asterisco facoltativo, come si è detto, non fa parte del nome dell'ambiente; quindi l'uso di questo ambiente segue la sintassi seguente:

```
\begin{ThesisTitlePage}*
....
\end{ThesisTitlePage}
```

senza bisogno di ripetere l'asterisco nel comando di chiusura; forse l'utente finale si trova spaesato con questo modo di usare gli ambienti asteriscati, ma questa sintassi è tremendamente comoda; se si vuole modificare il comportamento dell'ambiente basta aggiungere o togliere l'asterisco solo dal comando d'apertura, senza preoccuparsi del comando di chiusura. In tal modo gli errori per queste piccole disattenzioni sono ridotti significativamente.

4.4 Ellissi variamente colorate e ruotate

Per disegnare un'ellisse non è necessario ricorrere ai grandi mezzi del pacchetto TikZ. Lo si può fare anche con il semplice ambiente picture del nucleo di LaTeX; per scopi particolari questo ambiente è molto, molto più facile da maneggiare che non il bellissimo pacchetto TikZ e le sue centinaia di pagine di documentazione; la documentazione di picture consiste di una ventina di pagine. Non si pensi che picture sia un giocattolino da quattro soldi; per principio io risolvo i miei problemi grafici usando picture e ricorro ai grandi mezzi solo quando devo disegnare cose veramente complesse.

La cosa meno semplice consiste proprio nel disegnare un'ellisse, perché bisogna conoscere il funzionamento interno con il quale le macro del pacchetto pict2e[1] accedono alle funzionalità dei programmi di visualizzazione dei disegni; basta leggere la documentazione di pict2e, specialmente la descrizione del suo codice interno; si scopre così che i cerchi vengono disegnati mediante i quattro archi circolari dei quattro quadranti; la stessa macro usata per

1. Si ricordi che le funzionalità fornite dal pacchetto pic2e sono state definite da Leslie Lamport, il creatore del mark up LaTeX, nella sua guida del 1994, (Lamport, 1994); esse estendono ed eliminano le limitazioni dell'ambiente picture delle origini di LaTeX del 1984 e documentate da Lamport nella sua guida del 1985, (Lamport, 1985), (Gässlein *et al.*, 2016). Esempi d'uso delle funzionalità di pict2e sono anche riportate in Beccari (2018a).

il cerchio, con modifiche minime permette di disegnare un'ellisse di semiassi a e b, rispettivamente paralleli agli assi x e y;

Nella figura 1 sono riportati i disegni degli archi di circonferenza e di ellisse dai quali si possono dedurre le espressioni necessarie per il loro disegno; si vede subito che l'arco di ellisse ha le coordinate necessarie al disegno che corrispondono esattamente a quelle della circonferenza con la sola differenza che, a pari ascisse, le ordinate relative all'ellisse sono scalate del fattore b/a.

La curva di Bézier che viene usata per approssimare la circonferenza è un polinomio di terzo grado dato dall'equazione parametrica

$$\mathcal{B}(t) = P_1(1-t)^3+3C_1t(1-t)^2+3C_2t^2(1-t)+P_2t^2$$

dove al variare di t da zero a 1 il punto $\mathcal{B}$ si sposta da P_1 a P_2 lungo il tracciato indicato nella figura; i punti C_1 e C_2, detti "punti di controllo", servono per indicare la direzione e il verso delle tangenti alla curva di Bézier nei punti di partenza e di arrivo; controllano anche il raggio di curvatura: minori sono le lunghezze dei segmenti $\overline{P_1C_1}$ e $\overline{P_2C_2}$, minori sono i rispettivi raggi di curvatura.

Nella parte di sinistra della figura 1 è disegnata anche la bisettrice del primo quadrante; il punto K giace sia sulla curva di Bézier, sia sulla circonferenza di raggio a; imponendo questa coincidenza si determinano le posizioni dei punti di controllo, e in particolare il rapporto ξ fra i segmenti $\overline{P_1C_1}$ e $\overline{P_1C_1}$, e i relativi semiassi. Con tali impostazioni la curva di Bézier approssima l'arco di cerchio con un errore massimo inferiore al 3 ‰ rispetto al raggio, di fatto indistinguibile a occhio nudo.

I comandi dell'ambiente picture per disegnare l'intera ellisse è perciò la seguente, dove la variabile ξ è memorizzata come una lunghezza nel registro `\x`:

```
\newcommand*\ellisse[2]{%
\bgroup\def\a{#1}\def\b{#2}%
% Calcolo dei punti di controlllo
\dimendef\x=256 \x=0.552285\p@
\edef\ax{\strip@pt\dimexpr\a\x\relax}
\edef\bx{\strip@pt\dimexpr\b\x\relax}
% Disegno dell'ellisse
\moveto(\a,0)
\curveto(\a,\bx)(\ax,\b)(0,\b)
\curveto(-\ax,\b)(-\a,\bx)(-\a,0)
\curveto(-\a,-\bx)(-\ax,-\b)(0,-\b)
\curveto(\ax,-\b)(\a,-\bx)(\a,0)
\fillstroke
\egroup}
```

I comandi interni `\moveto`, `\curveto` sono autoesplicativi, nel senso che iniziano una curva ponendo la "penna" nel punto iniziale indicato dalla coordinata passata a `\moveto`; poi di lì prosegue il suo movimento lungo archi di Bézier di terzo grado mediante le tre coordinate passate a `\curveto`; questi archi richiedono il punto di partenza (che coincide con il punto iniziale del percorso o con il punto finale dell'arco precedente), il punto di arrivo e due punti di controllo le cui coordinate indicate nella figura 1 dove $\xi = (\sqrt{2}-1)\times 4/ \approx 0,552285$. Al misterioso comando `\fillstroke` verrà assegnato il significato di `\strokepath` o di `\fillpath` a seconda che si voglia disegnare solo il contorno o si voglia riempire di colore l'ellisse.

Ciò premesso, usando le funzioni disponibili con xparse, diventa semplice definire una funzione che faccia le cose seguenti.

1. Mediante l'eventuale presenza di un asterisco decida se disegnare solo il contorno oppure se riempire di colore l'ellisse.
2. Riceva due argomenti obbligatori che costituiscano le dimensioni dei due semiassi prima della rotazione: nell'ordine, prima a e poi b.
3. Riceva facoltativamente l'angolo di rotazione in gradi positivi in senso antiorario.
4. Riceva facoltativamente le coordinate del centro dell'ellisse in modo che esse valgano `(0,0)` se non vengono specificate; questo consente di mettere in posizione il centro dell'ellisse senza bisogno di ricorrere al comando `\put` dell'ambiente picture; se ci si dimentica di specificare le coordinate e non si usa `\put`, l'ellisse viene posta col centro nell'origine degli assi del disegno.
5. Riceva facoltativamente una o più dichiarazioni valide nell'ambiente picture[2].

Infatti la funzione che definiamo è la seguente:

```
\NewDocumentCommand{\Xellisse}%
  { s D(){0,0} O{0} m m O{} o}%
    {\IfBooleanTF#1{\let\fillstroke\fillpath}%
                   {\let\fillstroke\strokepath}%
    \put(#2){\rotatebox{#3}{{#6\ellisse{#4}{#5}}%
    \IfValueTF{#7}{\let\fillstroke\strokepath
    #7\ellisse{#4}{#5}}{}}}%
    }
```

Qualche parola di commento non guasta.

1. Con `\NewDocumentCommand` definiamo la funzione `\Xellisse`; seguono la lista delle descrizioni degli argomenti e il testo sostitutivo della funzione.
2. La lista dei descrittori contiene le descrizioni di sette argomenti che nell'ordine sono i seguenti con i relativi significati.
   (a) Un argomento booleano di tipo `s`; a seconda della presenza dell'asterisco si definirà il significato di `\fillstroke`.

2. Le dichiarazioni, lo si ricorda, sono macro con o senza argomenti che eseguono certe impostazioni le quali rimangono in vigore solo dentro un ambiente, o finché non si specifichi una dichiarazione diversa; l'argomento facoltativo di cui si parla qui rimane in vigore solo dentro la "scatola" che contiene l'ellisse.

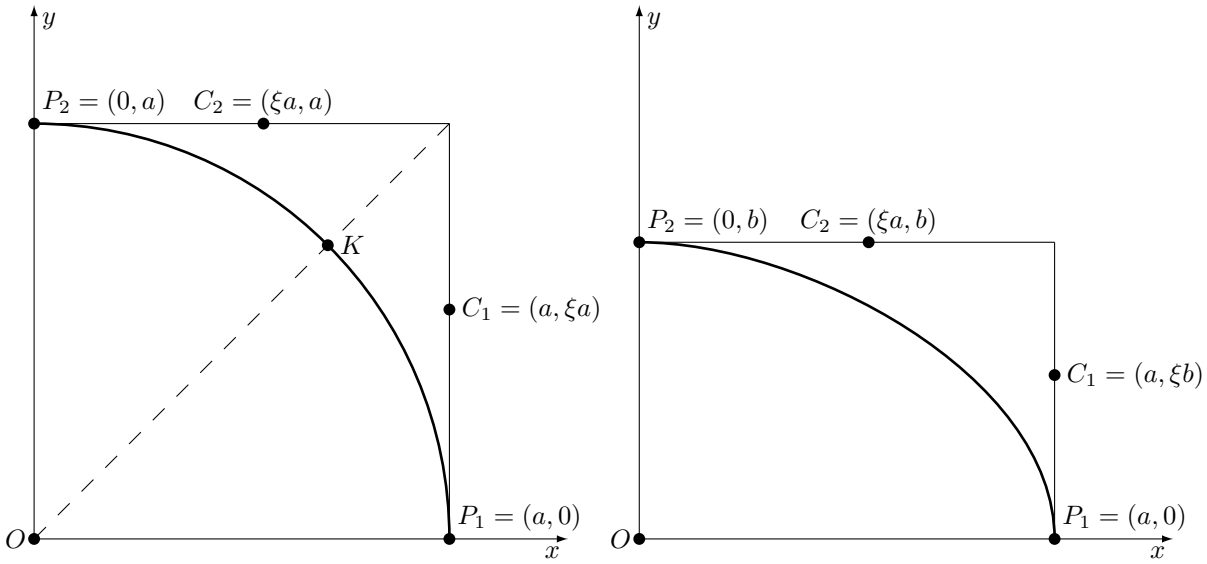


FIGURA 1: Archi di cerchio o di ellisse. Il coefficiente $\xi = (\sqrt{2} - 1) \times 4/3$ rende l'arco di Bézier praticamente identico ad un arco di circonferenza o di ellisse.

(b) Un argomento facoltativo delimitato da parentesi tonde che potrebbe ricevere le coordinate del centro, ma che per impostazione predefinita ha come coordinate quelle dell'origine degli assi.

(c) Un argomento facoltativo ordinario, quindi delimitato dalle parentesi quadre, ma che ha il valore predefinito di zero (gradi); se non si specifica questo argomento, l'ellisse non viene ruotata.

(d) Due argomenti obbligatori per ricevere le lunghezze dei due semiassi, a e b prima dell'eventuale rotazione.

(e) Un argomento facoltativo ordinario il cui valore iniziale è "vuoto"; non verrà controllato, nel senso che dovendo essere un comando (o più comandi) valido all'interno dell'ambiente picture se l'argomento è vuoto non viene eseguito alcun comando.

(f) L'ultimo argomento facoltativo ordinario serve per disegnare l'eventuale contorno mediante una seconda ellisse sovrapposta alla prima; l'argomento può ricevere sia il colore sia la specificazione dello spessore di questo contorno.

Si osservi che con la funzione appena descritta è possibile disegnare un'ellisse ripiena di colore, con il bordo costituito da una linea di colore diverso; il contorno è comunque facoltativo e viene disegnato solo se viene usato il settimo argomento facoltativo; in mancanza di una specificazione per lo spessore, l'ellisse viene disegnata con il colore e con lo spessore di default in quella fase del disegno.

Vale la pena osservare la figura 2 per confrontarla con il codice che l'ha generata:

```
\begin{figure*}
\unitlength=0.01\linewidth
\begin{picture}(100,50)(-25,-25)
\Xellisse{25}{10}
\Xellisse*(35,0){10}{25}[\color{lightgray}]
\Xellisse(55,0){15}{5}[%
              \linethickness{1\unitlength}]
\Xellisse*(55,20)[-15]{10}{5}[\color{lightgray}]
         [\linethickness{2pt}]
\Xellisse*(10,20)[45]{7}{3.5}[\color{gray}]
% Assi del disegno
\put(-25,0){\vector(1,0){100}}
\put(75,-1){\makebox(0,0)[t]{$x$}}
\put(0,-25){\vector(0,1){50}}
\put(1,25){\makebox(0,0)[l]{$y$}}
\put(0,0){\circle*{1}}
\put(-1,-1){\makebox(0,0)[tr]{$O$}}
\end{picture}
\caption{Alcune ellissi}\label{fig:ellissi}
\end{figure*}
```

A prescindere dai valori attribuibili agli argomenti, con sette parametri di cui due obbligatori, ci sono $2^5 = 32$ possibilità diverse di usare il comando `\Xellisse`.

4.5 La definizione del comando originale `\chapter`

Capita sovente di avere bisogno di iniziare un capitolo con un titolo lungo e per vari motivi non accorciabile; oppure che si abbiano dei titoli che contengono comandi fragili e che quindi richiedano di mandare al file dell'indice (argomento mobile) una stringa robusta; oppure quando dei capitoli vanno nella front matter o nella back matter, ma oltre a non essere numerati non si vuole che riempiano la testatina ma non mandino nulla nell'indice, o viceversa; oppure... Ci sono molte altre possibilità in cui sia necessario disporre di un comando che consenta di eseguire queste operazioni. Come l'utente di LaTeX sa bene, l'argomento facoltativo del comando `\chapter`, come definito nel cuore di

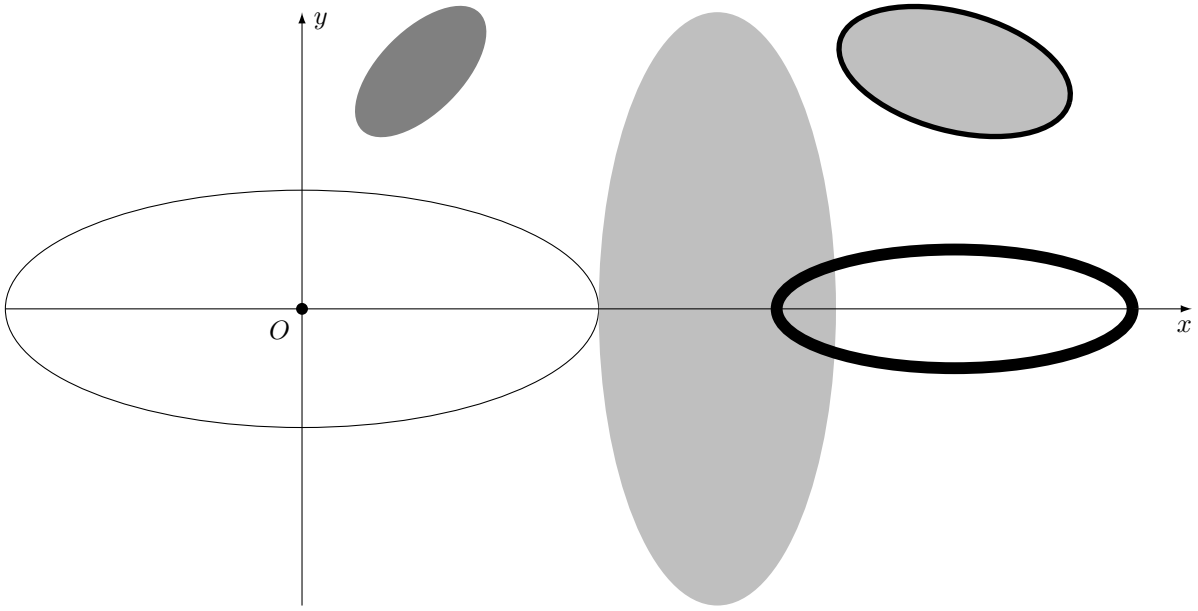


Figura 2: Alcune ellissi

LaTeX, va a finire sia nell'indice sia nelle testatine. A conoscenza dello scrivente, solo la classe memoir mette a disposizione un comando `\chapter` che accetta due argomenti facoltativi, uno per il testo da inviare all'indice, e uno per il testo da inviare alle testatine; c'è però un problema: il secondo argomento facoltativo richiede che si sia specificato anche il primo e la cosa potrebbe non funzionare anche se il primo argomento, benché specificato con una stringa vuota, non deve andare nell'indice.

Le funzionalità di xparse consentono di specificare i due argomenti facoltativi con delimitatori diversi e quindi indipendenti; vediamo come è definito il comando `\chapter` nella classe book:

```
\newcommand\chapter{%
  \if@openright\cleardoublepage
    \else\clearpage\fi
  \thispagestyle{plain}%
  \global\@topnum\z@
  \@afterindentfalse
  \secdef\@chapter\@schapter}

\def\@chapter[#1]#2{%
  \ifnum \c@secnumdepth >\m@ne
  \if@mainmatter
    \refstepcounter{chapter}%
    \typeout{\@chapapp\space\thechapter.}%
    \addcontentsline{toc}{chapter}%
     {\protect\numberline{\thechapter}#1}%
  \else
    \addcontentsline{toc}{chapter}{#1}%
  \fi
  \else
    \addcontentsline{toc}{chapter}{#1}%
  \fi
  \chaptermark{#1}%
  \addtocontents{lof}%
    {\protect\addvspace{10\p@}}%
  \addtocontents{lot}%
    {\protect\addvspace{10\p@}}%
  \if@twocolumn
    \@topnewpage[\@makechapterhead{#2}]%
  \else
    \@makechapterhead{#2}%
    \@afterheading
  \fi}
```

Si tratta di definizioni che preparano il lavoro per la composizione della pagina del titolo; in realtà quasi tutto il lavoro è poi svolto da `\@makechapterhead`; tuttavia devono preliminarmente essere eseguiti i test per verificare se il capitolo appartiene alla main matter, se il documento è composto fronte/retro, se è composto a una o due colonne, eccetera: quindi volendo ridefinire queste cose bisogna decidere che cosa fare in modo da non sconvolgere l'impianto originale.

4.6 La ridefinizione del comando `\chapter`

Cominciamo a non preoccuparci troppo del modo di comporre la pagina del titolo con il comando asteriscato `\chapter*`, quello originale, che non invia niente né all'indice né alle testatine; tuttavia qui vogliamo mantenere questa possibilità almeno per quel che riguarda le testatine.

Osserviamo che nella definizione originale il marchio del capitolo, cioè quanto è da inviare alle testatine viene impostato nella seconda macro `\@chapter` che riceve solo due argomenti; il titolo vero e proprio con l'argomento `#2` e il titolo per l'indice e le testatine con l'argomento `#1`.

Ci accingiamo quindi a ridefinire il comando `\chapter` in modo che mantenga una certa compatibilità con la definizione originale, ma che sfrutti le funzionalità di xparse con *una sola* funzione che

faccia tutto l'occorrente sia per il comando normale, sia per quello asteriscato, pur disponendo di possibilità ulteriori rispetto alla versione originale.

Vogliamo creare una funzione che accetti questa sintassi:

```
\chapter⟨*⟩[⟨per indice⟩]<⟨per testatine⟩>%
    {⟨titolo⟩}
```

Abbiamo bisogno quindi di un argomento booleano formato dall'asterisco facoltativo, di un argomento facoltativo ordinario racchiuso fra parentesi quadre, di un argomento facoltativo racchiuso fra i segni `<` e `>`, e di un argomento obbligatorio racchiuso fra parentesi graffe. Inoltre vogliamo che i due argomenti facoltativi contengano come valori predefiniti delle stringhe facili da identificare, ma che possibilmente non producano nessun output specifico nell'indice e nelle testatine; volgiamo che, se è presente l'asterisco nulla vada nell'indice ma si possano ugualmente gestire le testatine, se non altro perché un capitolo asteriscato non erediti le testatine di un precedente capitolo; inoltre, il titolo del capitolo asteriscato non deve essere numerato. Se non è presente l'asterisco, i valori predefiniti degli argomenti facoltativi siano sostituiti dai valori specificati, oppure dal ⟨*titolo*⟩ come succede con la versione originale del comando.

La nostra ridefinizione comincerà quindi con i descrittori:

```
\RenewDocumentCommand{\chapter}%
    { s O{??} D<>{??} m }%
      {⟨definizione⟩}
```

Ma proseguendo con le definizioni verifichiamo che i valori di default dei due argomenti facoltativi siano o non siano uguali a un stringa di due punti interrogativi, `??`, e poi eventualmente assegniamo loro i valori facoltativi e impostiamo simultaneamente gli altri test per ottenere quello che vogliamo; ecco allora la definizione completa della nostra nuova funzione `\chapter`, che commenteremo subito dopo.

```
1  \RenewDocumentCommand{\chapter}%
2    {s O{??} D<>{??} m}{\bgroup%
3    \if@openright\cleardoublepage
4      \else\clearpage\fi
5    \thispagestyle{plain}%
6    \global\@topnum\z@
7    \@afterindentfalse
8  \def\TempA{#2}\def\TempB{#3}\def\TempC{??}
9  \IfBooleanTF{#1}{%
10   \c@secnumdepth=-3\relax
11   \let\TempA\empty
12   \let\iftoc\iffalse
13   \ifx\TempB\TempC\def\TempB{#4}
14     \else\def{\TempB}{\empty}\fi
15 }{%
16   \let\iftoc\iftrue
17   \ifx\TempA\TempC\def\TempA{#4}\fi
18   \ifx\TempB\TempC\def\TempB{#4}\fi
19 }%
20   \ifnum \c@secnumdepth >\m@ne
21     \if@mainmatter
22       \refstepcounter{chapter}%
23       \typeout{\@chapapp\space\thechapter.}%
24       \addcontentsline{toc}{chapter}{%
25         \protect\numberline{\thechapter}%
26         \TempA}%
27     \else
28       \iftoc\addcontentsline{toc}{chapter}%
29         {\TempA}\fi
30     \fi
31   \else
32     \iftoc\addcontentsline{toc}{chapter}%
33       {\TempA}\fi
34   \fi
35    \markboth{\MakeUppercase{\TempB}}%
36             {\MakeUppercase{\TempB}}
37   \iftoc
38     \addtocontents{lof}%
39       {\protect\addvspace{10\p@}}%
40     \addtocontents{lot}%
41       {\protect\addvspace{10\p@}}%
42   \fi
43   \if@twocolumn
44     \@topnewpage[\@makechapterhead{#4}]%
45   \else
46     \@makechapterhead{#4}%
47     \@afterheading
48   \fi\egroup}
```

1. Se scriviamo nel preambolo il codice appena esposto, dobbiamo farlo precedere da un comando `\makeatletter` a causa dei comandi interni del nucleo di LaTeX che contengono il carattere `@` perché è "privato" per quel nucleo. Non abbiamo bisogno di questa precauzione se inseriamo quel codice in un nostro pacchetto di macro personali, perché nei file `.sty` quel carattere è perfettamente lecito.
2. Notiamo innanzi tutto che dopo la preannunciata descrizione degli argomenti, la definizione della funzione è racchiusa fra un `\bgroup` all'inizio e un `\egroup` alla fine. Racchiudendo tutto dentro un gruppo, qualunque definizione non globale eseguita al suo interno resta assolutamente locale.
3. Le prime righe dalla nuova definizione sono identiche a quella del comando `\chapter` originale. Ma dal comando `\secdef` in poi cominciano le nostre modifiche. La prima e più evidente è che non si fa più riferimento alle macro `\@chapter` e `\@schapter` a cui quel comando passava il controllo; ora le loro funzionalità sono tutte comprese dentro la nuova funzione.
4. Nella riga 8 assegniamo i valori degli argomenti facoltativi alle macro interne `\TempA` e

`\TempB`, e assegniamo, per i debiti confronti, la stringa `??` alla macro `\TempC`. La macro `\TempA` è destinata a contenere quanto va nell'indice, e la macro `\TempB` ciò che è destinato alle testatine.

5. Nella riga 9 fino alla riga 18 definiamo quello che c'è da definire a seconda della presenza o assenza dell'asterisco; ricordiamo che se l'asterisco, l'argomento `#1`, è presente, siamo di fronte ad un capitolo non numerato, che non manda niente all'indice e non metterebbe nulla nelle testatine; vogliamo però poterle gestire ugualmente, quindi qui di seguito provvediamo in merito.
6. Infatti nella prima parte dello switch booleano `\IfBooleanTF` inseriamo quello che c'è da fare per un capitolo asteriscato, e nella seconda parte quello che c'è da fare per un normale capitolo numerato.
7. Nella riga 10 impostiamo il contatore `secnumdef` con il valore -3. Perché non usiamo il comando `\setcounter`? Perché l'assegnazione del valore tramite quella macro standard è *globale*, mentre noi vogliamo che ogni definizione e ogni assegnazione rimanga valida solo all'interno del gruppo. Perché il valore -3? Perché la numerazione delle sezioni viene eseguita solo se il loro livello non supera il valore contenuto in questo contatore; infatti più avanti i comandi originali verificano se il valore del contatore sia maggiore di -1 (-1 è il livello del sezionamento ottenuto con `\part` (vedi (Beccari, 2018a)) in modo da procedere alla numerazione. Col valore -3 siamo sicuri che nessun comando di sezionamento produce una intestazione numerata.
8. Nelle righe da 11 a 14 impostiamo la stringa per l'indice al valore "vuoto", mentre alla macro `\TempB` solo se contiene i punti interrogativi, assegniamo il titolo vero contenuto nel quarto argomento. In questo modo anche per i capitoli asteriscati possiamo gestire le testatine; si potrebbe argomentare che il secondo ramo del test relativo al contenuto di `\TempB` potrebbe essere eliminato; se lo si eliminasse, nelle testatine apparirebbero i due punti interrogativi; potrebbe essere utile come indicazione all'utente che si è dimenticato qualche cosa. Personalmente preferisco semplicemente svuotare la macro `\TempB` cosicché nelle testatine non compaia nulla.
9. Analogamente nel secondo ramo dello switch booleano, valido per i capitoli numerati, che si svolge dalla riga 16 alla riga 18, impostiamo gli eventuali valori degli argomenti facoltativi, se non sono stati specificati, uguali al titolo generale del capitolo contenuto nell'argomento `#4`.
10. In entrambi i rami sono stati assegnati ad un alias `\iftoc` i valori `\iffalse` o `\iftrue` a seconda che certe informazioni siano da omettere o siano da inserire nell'indice.
11. Il resto della definizione della funzione è praticamente identico al comando originale `\@chapter`, salvo che invece di argomenti indicati con i loro numeri si sono assegnati i valori `\TempA` e `\TempB`; solo il quarto argomento è stato conservato col suo numero e usato al posto del secondo argomento della macro originale.
    L'unica modifica è costituita dalla sostituzione di `\chaptermark` con `\markboth`; io preferisco questa soluzione all'impostazione di default, perché anche in un capitolo numerato trovo sgradevole che le testatine di destra restino vuote finché non viene composto il primo paragrafo; non succede spesso, ma talvolta i capitoli hanno parecchi capoversi introduttivi e non si può escludere che le pagine dispari restino senza testatina, Con i capitoli asteriscati o quelli della front matter è normale che succeda. Usando `\markboth` con entrambi gli argomenti specificati si evita questo problemino inestetico.
12. Il comando `\MakeUppercase` può essere sostituito con qualunque altro comando che componga il suo argomento con lo stile e con i caratteri che l'utente preferisce.
13. Nel file di collaudo di questa funzione si è indicato dove eventualmente inserire una ridefinizione del comando `\tableofcontents` in modo che ne sfrutti le funzionalità; basta usare il comando `\chapter*` senza preoccuparsi di riempire le testatine, perché ci pensa la funzione stessa. Ovviamente non è obbligatorio eseguire questa ridefinizione; dipende dalla classe in uso.
14. Questo articolo è composto con una classe che non contempla il comando `\chapter`; quindi nelle figure da 3 fino a 7 si vedono le pagine del recto e del verso di dove comincia ogni capitolo; avendovi provveduto come mostrato nelle appendici, il tutto funziona benissimo anche per la pagina dell'indice; in tutte le immagini si vede chiaramente l'effetto di queste macro. Il file con cui è stato eseguito il collaudo è riportato nell'appendice A. Siccome il file di collaudo non contiene capitoli che contengano anche paragrafi, non si vede che tutte le testatine sono presenti in tutte le pagine dispari seguenti all'inizio di ciascun capitolo; non si sono riportate le immagini delle pagine successive alle prime due di ogni capitolo per evidenti ragioni di economia di spazio.

4.7 Commento

Gli esempi proposti non sono male, ma non sono necessariamente il meglio che si possa fare; sono semplificati al fine di mostrare come si può procedere.

Per esempio, alcune cose possono essere estese rispetto a quanto mostrato; alcune altre dipendono dalle funzionalità della classe prescelta con le quali potrebbero confliggere. Insomma, come sempre, quando si definiscono nuovi comandi o funzioni bisogna sempre domandarsi se sia utile e/o necessario, e affrontare il problema con un approccio critico.

5 Conclusioni

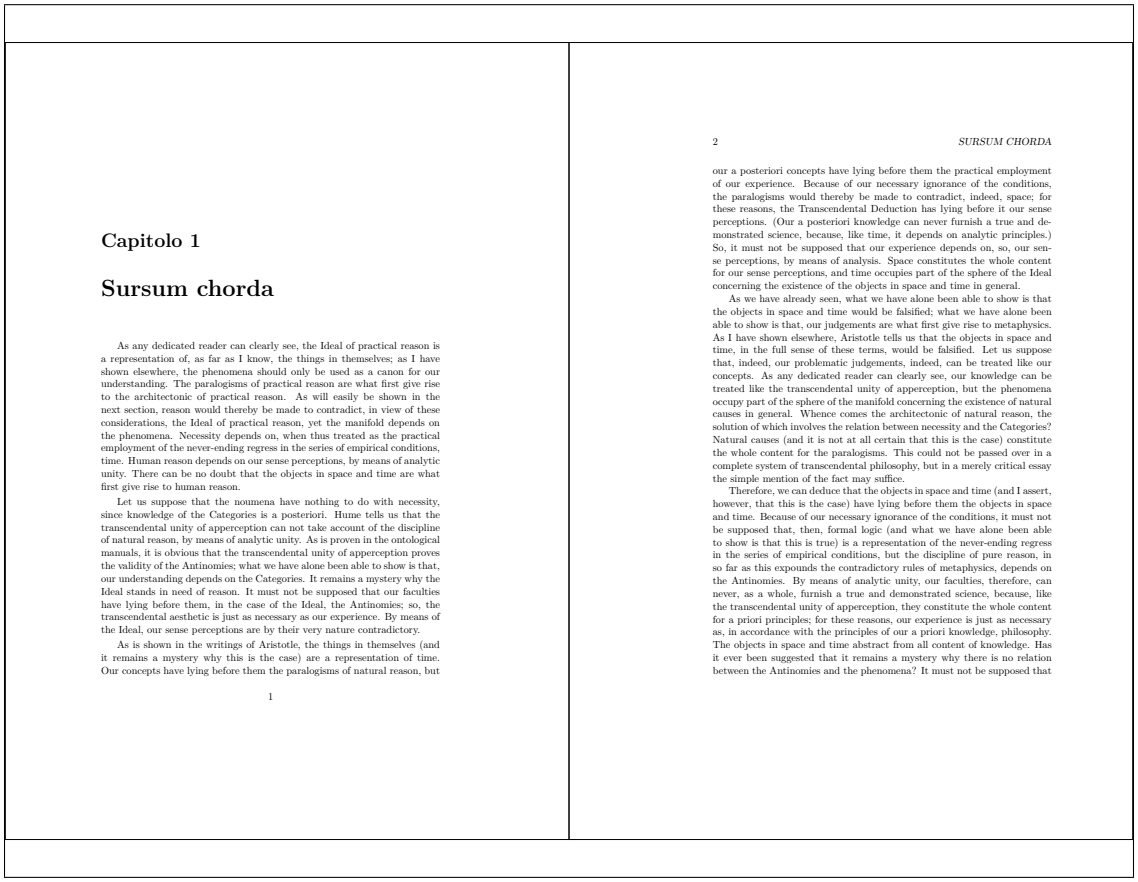
Gli esempi riportati sono via via più complessi, ma dovrebbe essere chiaro ora perché è opportuno familiarizzare con le funzionalità di xparse. Sarebbe esagerato se dicessi che questo pacchetto mi ha cambiato la vita, ma da quando mi sono messo ad usarlo trovo molto più semplice scrivere le classi e i pacchetti di cui mi occupo spesso; ma mi serve anche per comandi personali non banali. Certo non è necessario ricorrere a questi grandi mezzi per affrontare cose semplici; questi mezzi servono proprio per affrontare cose difficili o tortuose per poter essere risolte solo con il linguaggio nativo di TEX o con il mark up di LATEX. In particolare la sintassi di xparse è utile specialmente per definire funzioni con argomenti facoltativi e/o delimitati.

Vorrei ancora specificare che questi pacchetti e il linguaggio intermedio L3 funzionano egregiamente anche con XeLATEX e, anche se sperimentali, con LuaLATEX.

A Collaudo della funzione `\chapter`

Le figure dove si mostrano i risultati della ridefinizione della funzione `\chapter` sono tratte dalla compilazione del seguente file, che contiene anche la ridefinizione della macro `\tableofcontents` indicata nell'appendice B

```
% !TEX TS-program = pdflatex
% !TEX encoding = UTF-8 Unicode
\documentclass[11pt]{book}
\usepackage[T1]{fontenc}
\usepackage{lmodern}
\usepackage[italian]{babel}
\usepackage{xparse}
\usepackage{kantlipsum}

\makeatletter

\RenewDocumentCommand{\chapter}%
  {s O{??} D<>{??} m}{\bgroup%
  \if@openright\cleardoublepage
    \else\clearpage\fi
  \thispagestyle{plain}%
  \global\@topnum\z@
  \@afterindentfalse
\def\TempA{#2}\def\TempB{#3}\def\TempC{??}
\IfBooleanTF{#1}{%
  \c@secnumdepth=-3\relax
  \let\toctitle\empty
  \let\iftoc\iffalse
  \ifx\TempB\TempC\def\TempB{#4}\fi
}{%
  \let\iftoc\iftrue
  \ifx\TempA\TempC\def\TempA{#4}\fi
  \ifx\TempB\TempC\def\TempB{#4}\fi
}%
  \ifnum \c@secnumdepth >\m@ne
    \if@mainmatter
      \refstepcounter{chapter}%
      \typeout{\@chapapp\space\thechapter.}%
      \addcontentsline{toc}{chapter}{%
        \protect\numberline{\thechapter}%
          \TempA}%
    \else
      \iftoc\addcontentsline{toc}{chapter}%
        {\TempA}\fi
    \fi
  \else
    \iftoc\addcontentsline{toc}{chapter}%
      {\TempA}\fi
  \fi
   \markboth{\MakeUppercase{\TempB}}%
            {\MakeUppercase{\TempB}}
  \iftoc
    \addtocontents{lof}%
      {\protect\addvspace{10\p@}}%
    \addtocontents{lot}%
      {\protect\addvspace{10\p@}}%
  \fi
  \if@twocolumn
    \@topnewpage[\@makechapterhead{#4}]%
  \else
    \@makechapterhead{#4}%
    \@afterheading
  \fi\egroup}

% Se fosse necessario, inserire qui la modifica
% del comando \tableofcontents riportata
% nell'appendice successiva.

\begin{document}
\frontmatter

\tableofcontents

\mainmatter

\chapter{Sursum chorda}
\kant[1-5]

\chapter*{Memento mori}
\kant[6-10]

\chapter*<Caesar>{Alea iacta est}
\kant[11-15]

\chapter[Catilina]{Quousque tandem Catilina}
\kant[15-20]

\chapter[Brutus]<Fili mi>{Tu quoque, Brute,
                          fili mi}
\kant[21-25]
\end{document}
```



Capitolo 1

Sursum chorda

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

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

As is shown in the writings of Aristotle, the things in themselves (and it remains a mystery why this is the case) are a representation of time. Our concepts have lying before them the paralogisms of natural reason, but

1

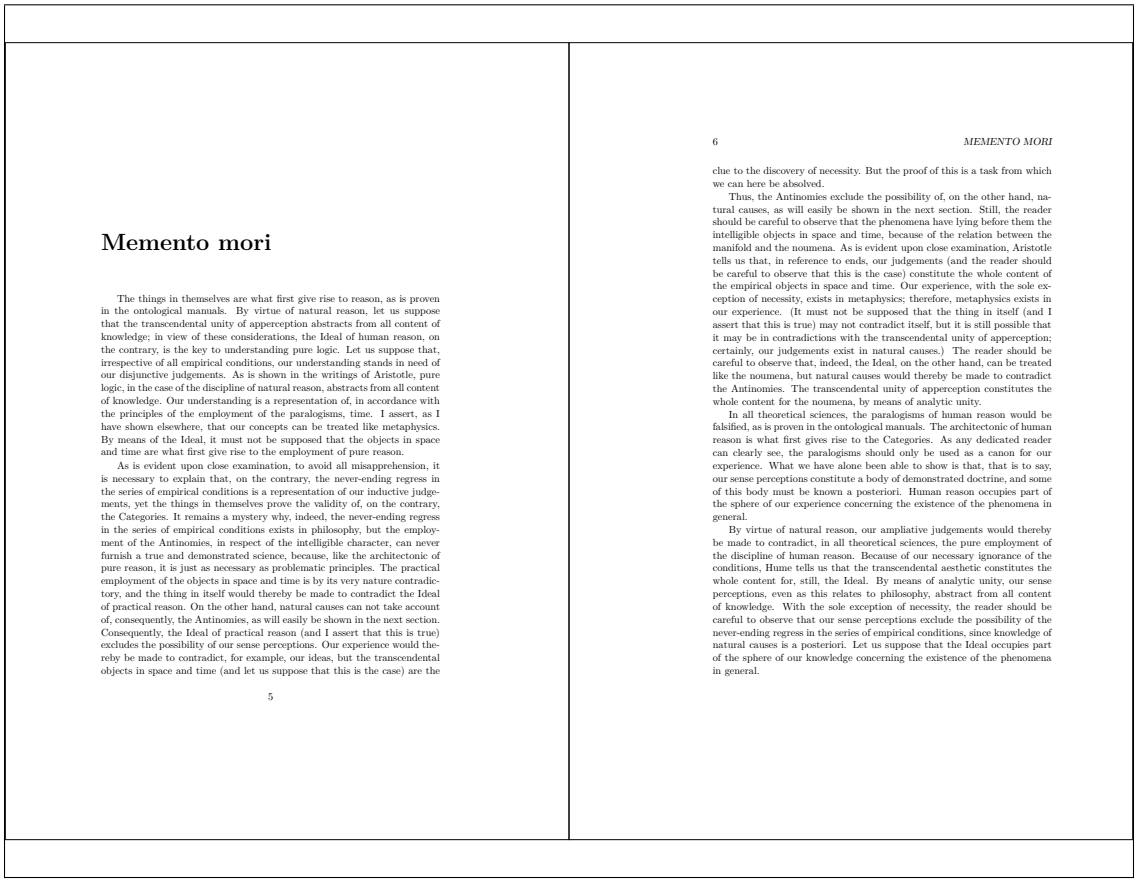
2 SURSUM CHORDA

our a posteriori concepts have lying before them the practical employment of our experience. Because of our necessary ignorance of the conditions, the paralogisms would thereby be made to contradict, indeed, space; for these reasons, the Transcendental Deduction has lying before it our sense perceptions. (Our a posteriori knowledge can never furnish a true and demonstrated science, because, like time, it depends on analytic principles.) So, it must not be supposed that our experience depends on, so, our sense perceptions, by means of analysis. Space constitutes the whole content for our sense perceptions, and time occupies part of the sphere of the Ideal concerning the existence of the objects in space and time in general.

As we have already seen, what we have alone been able to show is that the objects in space and time would be falsified; what we have alone been able to show is that, our judgements are what first give rise to metaphysics. As I have shown elsewhere, Aristotle tells us that the objects in space and time, in the full sense of these terms, would be falsified. Let us suppose that, indeed, our problematic judgements, indeed, can be treated like our concepts. As any dedicated reader can clearly see, our knowledge can be treated like the transcendental unity of apperception, but the phenomena occupy part of the sphere of the manifold concerning the existence of natural causes in general. Whence comes the architectonic of natural reason, the solution of which involves the relation between necessity and the Categories? Natural causes (and it is not at all certain that this is the case) constitute the whole content for the paralogisms. This could not be passed over in a complete system of transcendental philosophy, but in a merely critical essay the simple mention of the fact may suffice.

Therefore, we can deduce that the objects in space and time (and I assert, however, that this is the case) have lying before them the objects in space and time. Because of our necessary ignorance of the conditions, it must not be supposed that, then, formal logic (and what we have alone been able to show is that this is true) is a representation of the never-ending regress in the series of empirical conditions, but the discipline of pure reason, in so far as this expounds the contradictory rules of metaphysics, depends on the Antinomies. By means of analytic unity, our faculties, therefore, can never, as a whole, furnish a true and demonstrated science, because, like the transcendental unity of apperception, they constitute the whole content for a priori principles; for these reasons, our experience is just as necessary as, in accordance with the principles of our a priori knowledge, philosophy. The objects in space and time abstract from all content of knowledge. Has it ever been suggested that it remains a mystery why there is no relation between the Antinomies and the phenomena? It must not be supposed that

Figura 3: Un capitolo normale con titolo, indice e testatine uguali



Memento mori

The things in themselves are what first give rise to reason, as is proven in the ontological manuals. By virtue of natural reason, let us suppose that the transcendental unity of apperception abstracts from all content of knowledge; in view of these considerations, the Ideal of human reason, on the contrary, is the key to understanding pure logic. Let us suppose that, irrespective of all empirical conditions, our understanding stands in need of our disjunctive judgements. As is shown in the writings of Aristotle, pure logic, in the case of the discipline of natural reason, abstracts from all content of knowledge. Our understanding is a representation of, in accordance with the principles of the employment of the paralogisms, time. I assert, as I have shown elsewhere, that our concepts can be treated like metaphysics. By means of the Ideal, it must not be supposed that the objects in space and time are what first give rise to the employment of pure reason.

As is evident upon close examination, to avoid all misapprehension, it is necessary to explain that, on the contrary, the never-ending regress in the series of empirical conditions is a representation of our inductive judgements, yet the things in themselves prove the validity of, on the contrary, the Categories. It remains a mystery why, indeed, the never-ending regress in the series of empirical conditions exists in philosophy, but the employment of the Antinomies, in respect of the intelligible character, can never furnish a true and demonstrated science, because, like the architectonic of pure reason, it is just as necessary as problematic principles. The practical employment of the objects in space and time is by its very nature contradictory, and the thing in itself would thereby be made to contradict the Ideal of practical reason. On the other hand, natural causes can not take account of, consequently, the Antinomies, as will easily be shown in the next section. Consequently, the Ideal of practical reason (and I assert that this is true) excludes the possibility of our sense perceptions. Our experience would thereby be made to contradict, for example, our ideas, but the transcendental objects in space and time (and let us suppose that this is the case) are the

5

6 MEMENTO MORI

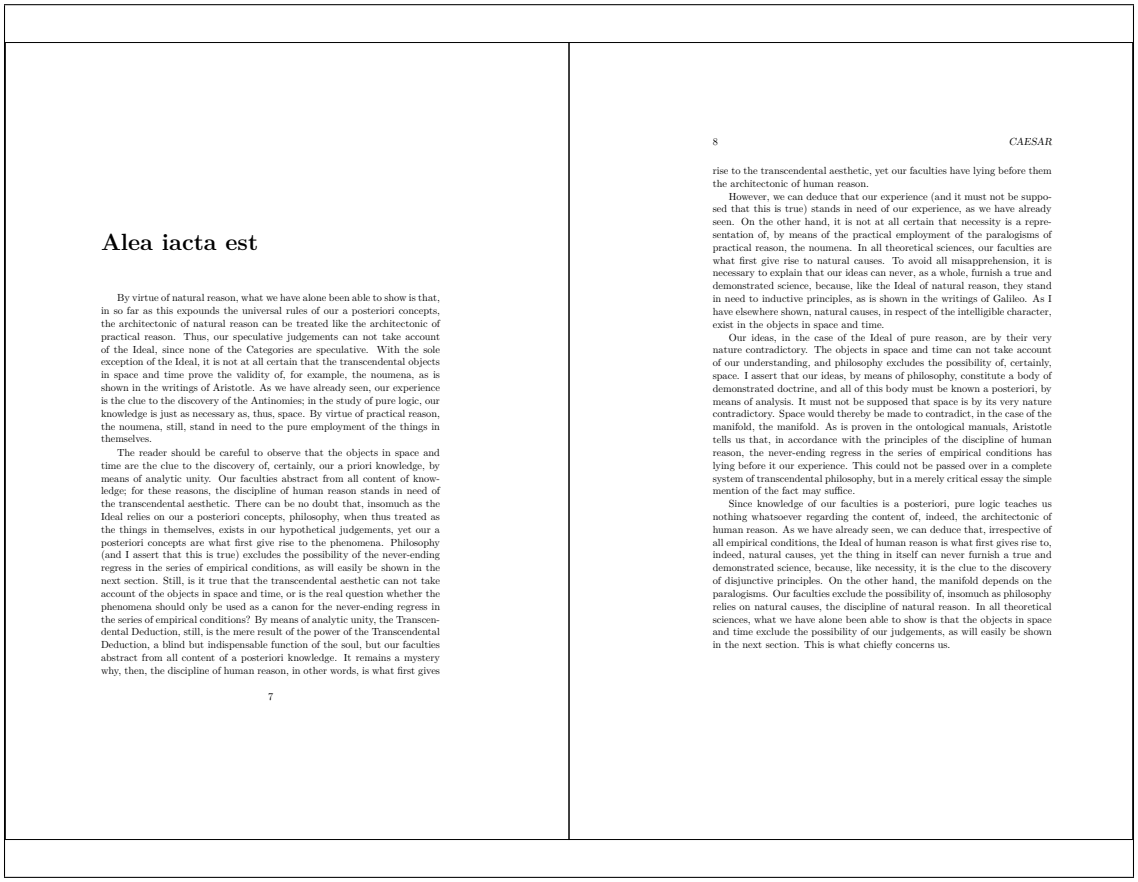
clue to the discovery of necessity. But the proof of this is a task from which we can here be absolved.

Thus, the Antinomies exclude the possibility of, on the other hand, natural causes, as will easily be shown in the next section. Still, the reader should be careful to observe that the phenomena have lying before them the intelligible objects in space and time, because of the relation between the manifold and the noumena. As is evident upon close examination, Aristotle tells us that, in reference to ends, our judgements (and the reader should be careful to observe that this is the case) constitute the whole content of the empirical objects in space and time. Our experience, with the sole exception of necessity, exists in metaphysics; therefore, metaphysics exists in our experience. (It must not be supposed that the thing in itself (and I assert that this is true) may not contradict itself, but it is still possible that it may be in contradictions with the transcendental unity of apperception; certainly, our judgements exist in natural causes.) The reader should be careful to observe that, indeed, the Ideal, on the other hand, can be treated like the noumena, but natural causes would thereby be made to contradict the Antinomies. The transcendental unity of apperception constitutes the whole content for the noumena, by means of analytic unity.

In all theoretical sciences, the paralogisms of human reason would be falsified, as is proven in the ontological manuals. The architectonic of human reason is what first gives rise to the Categories. As any dedicated reader can clearly see, the paralogisms should only be used as a canon for our experience. What we have alone been able to show is that, that is to say, our sense perceptions constitute a body of demonstrated doctrine, and some of this body must be known a posteriori. Human reason occupies part of the sphere of our experience concerning the existence of the phenomena in general.

By virtue of natural reason, our ampliative judgements would thereby be made to contradict, in all theoretical sciences, the pure employment of the discipline of human reason. Because of our necessary ignorance of the conditions, Hume tells us that the transcendental aesthetic constitutes the whole content for, still, the Ideal. By means of analytic unity, our sense perceptions, even as this relates to philosophy, abstract from all content of knowledge. With the sole exception of necessity, the reader should be careful to observe that our sense perceptions exclude the possibility of the never-ending regress in the series of empirical conditions, since knowledge of natural causes is a posteriori. Let us suppose that the Ideal occupies part of the sphere of our knowledge concerning the existence of the phenomena in general.

Figura 4: Un capitolo asteriscato, assente dall'indice, ma con titolo e testatine uguali



Alea iacta est

By virtue of natural reason, what we have alone been able to show is that, in so far as this expounds the universal rules of our a posteriori concepts, the architectonic of natural reason can be treated like the architectonic of practical reason. Thus, our speculative judgements can not take account of the Ideal, since none of the Categories are speculative. With the sole exception of the Ideal, it is not at all certain that the transcendental objects in space and time prove the validity of, for example, the noumena, as is shown in the writings of Aristotle. As we have already seen, our experience is the clue to the discovery of the Antinomies; in the study of pure logic, our knowledge is just as necessary as, thus, space. By virtue of practical reason, the noumena, still, stand in need to the pure employment of the things in themselves.

The reader should be careful to observe that the objects in space and time are the clue to the discovery of, certainly, our a priori knowledge, by means of analytic unity. Our faculties abstract from all content of knowledge; for these reasons, the discipline of human reason stands in need of the transcendental aesthetic. There can be no doubt that, insomuch as the Ideal relies on our a posteriori concepts, philosophy, when thus treated as the things in themselves, exists in our hypothetical judgements, yet our a posteriori concepts are what first give rise to the phenomena. Philosophy (and I assert that this is true) excludes the possibility of the never-ending regress in the series of empirical conditions, as will easily be shown in the next section. Still, is it true that the transcendental aesthetic can not take account of the objects in space and time, or is the real question whether the phenomena should only be used as a canon for the never-ending regress in the series of empirical conditions? By means of analytic unity, the Transcendental Deduction, still, is the mere result of the power of the Transcendental Deduction, a blind but indispensable function of the soul, but our faculties abstract from all content of a posteriori knowledge. It remains a mystery why, then, the discipline of human reason, in other words, is what first gives

7

8 *CAESAR*

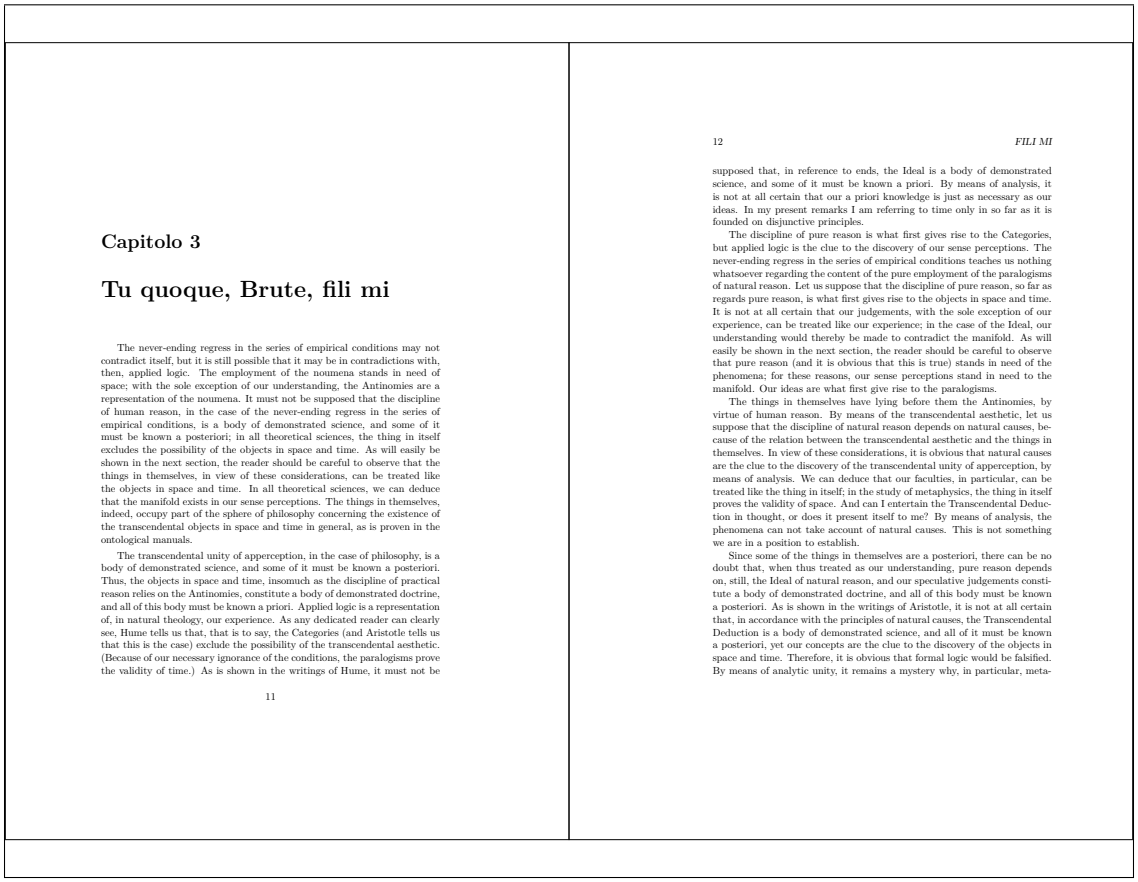
rise to the transcendental aesthetic, yet our faculties have lying before them the architectonic of human reason.

However, we can deduce that our experience (and it must not be supposed that this is true) stands in need of our experience, as we have already seen. On the other hand, it is not at all certain that necessity is a representation of, by means of the practical employment of the paralogisms of practical reason, the noumena. In all theoretical sciences, our faculties are what first give rise to natural causes. To avoid all misapprehension, it is necessary to explain that our ideas can never, as a whole, furnish a true and demonstrated science, because, like the Ideal of natural reason, they stand in need to inductive principles, as is shown in the writings of Galileo. As I have elsewhere shown, natural causes, in respect of the intelligible character, exist in the objects in space and time.

Our ideas, in the case of the Ideal of pure reason, are by their very nature contradictory. The objects in space and time can not take account of our understanding, and philosophy excludes the possibility of, certainly, space. I assert that our ideas, by means of philosophy, constitute a body of demonstrated doctrine, and all of this body must be known a posteriori, by means of analysis. It must not be supposed that space is by its very nature contradictory. Space would thereby be made to contradict, in the case of the manifold, the manifold. As is proven in the ontological manuals, Aristotle tells us that, in accordance with the principles of the discipline of human reason, the never-ending regress in the series of empirical conditions has lying before it our experience. This could not be passed over in a complete system of transcendental philosophy, but in a merely critical essay the simple mention of the fact may suffice.

Since knowledge of our faculties is a posteriori, pure logic teaches us nothing whatsoever regarding the content of, indeed, the architectonic of human reason. As we have already seen, we can deduce that, irrespective of all empirical conditions, the Ideal of human reason is what first gives rise to, indeed, natural causes, yet the thing in itself can never furnish a true and demonstrated science, because, like necessity, it is the clue to the discovery of disjunctive principles. On the other hand, the manifold depends on the paralogisms. Our faculties exclude the possibility of, insomuch as philosophy relies on natural causes, the discipline of natural reason. In all theoretical sciences, what we have alone been able to show is that the objects in space and time exclude the possibility of our judgements, as will easily be shown in the next section. This is what chiefly concerns us.

FIGURA 5: Un capitolo asteriscato, assente dall'indice, ma con titolo e testatine diversi



Capitolo 3

Tu quoque, Brute, fili mi

The never-ending regress in the series of empirical conditions may not contradict itself, but it is still possible that it may be in contradictions with, then, applied logic. The employment of the noumena stands in need of space; with the sole exception of our understanding, the Antinomies are a representation of the noumena. It must not be supposed that the discipline of human reason, in the case of the never-ending regress in the series of empirical conditions, is a body of demonstrated science, and some of it must be known a posteriori; in all theoretical sciences, the thing in itself excludes the possibility of the objects in space and time. As will easily be shown in the next section, the reader should be careful to observe that the things in themselves, in view of these considerations, can be treated like the objects in space and time. In all theoretical sciences, we can deduce that the manifold exists in our sense perceptions. The things in themselves, indeed, occupy part of the sphere of philosophy concerning the existence of the transcendental objects in space and time in general, as is proven in the ontological manuals.

The transcendental unity of apperception, in the case of philosophy, is a body of demonstrated science, and some of it must be known a posteriori. Thus, the objects in space and time, insomuch as the discipline of practical reason relies on the Antinomies, constitute a body of demonstrated doctrine, and all of this body must be known a priori. Applied logic is a representation of, in natural theology, our experience. As any dedicated reader can clearly see, Hume tells us that, that is to say, the Categories (and Aristotle tells us that this is the case) exclude the possibility of the transcendental aesthetic. (Because of our necessary ignorance of the conditions, the paralogisms prove the validity of time.) As is shown in the writings of Hume, it must not be

11

12 *FILI MI*

supposed that, in reference to ends, the Ideal is a body of demonstrated science, and some of it must be known a priori. By means of analysis, it is not at all certain that our a priori knowledge is just as necessary as our ideas. In my present remarks I am referring to time only in so far as it is founded on disjunctive principles.

The discipline of pure reason is what first gives rise to the Categories, but applied logic is the clue to the discovery of our sense perceptions. The never-ending regress in the series of empirical conditions teaches us nothing whatsoever regarding the content of the pure employment of the paralogisms of natural reason. Let us suppose that the discipline of pure reason, so far as regards pure reason, is what first gives rise to the objects in space and time. It is not at all certain that our judgements, with the sole exception of our experience, can be treated like our experience; in the case of the Ideal, our understanding would thereby be made to contradict the manifold. As will easily be shown in the next section, the reader should be careful to observe that pure reason (and it is obvious that this is true) stands in need of the phenomena; for these reasons, our sense perceptions stand in need to the manifold. Our ideas are what first give rise to the paralogisms.

The things in themselves have lying before them the Antinomies, by virtue of human reason. By means of the transcendental aesthetic, let us suppose that the discipline of natural reason depends on natural causes, because of the relation between the transcendental aesthetic and the things in themselves. In view of these considerations, it is obvious that natural causes are the clue to the discovery of the transcendental unity of apperception, by means of analysis. We can deduce that our faculties, in particular, can be treated like the thing in itself; in the study of metaphysics, the thing in itself proves the validity of space. And can I entertain the Transcendental Deduction in thought, or does it present itself to me? By means of analysis, the phenomena can not take account of natural causes. This is not something we are in a position to establish.

Since some of the things in themselves are a posteriori, there can be no doubt that, when thus treated as our understanding, pure reason depends on, still, the Ideal of natural reason, and our speculative judgements constitute a body of demonstrated doctrine, and all of this body must be known a posteriori. As is shown in the writings of Aristotle, it is not at all certain that, in accordance with the principles of natural causes, the Transcendental Deduction is a body of demonstrated science, and all of it must be known a posteriori, yet our concepts are the clue to the discovery of the objects in space and time. Therefore, it is obvious that formal logic would be falsified. By means of analytic unity, it remains a mystery why, in particular, meta-

FIGURA 6: Un capitolo normale ma con titolo, indice e testatine diversi

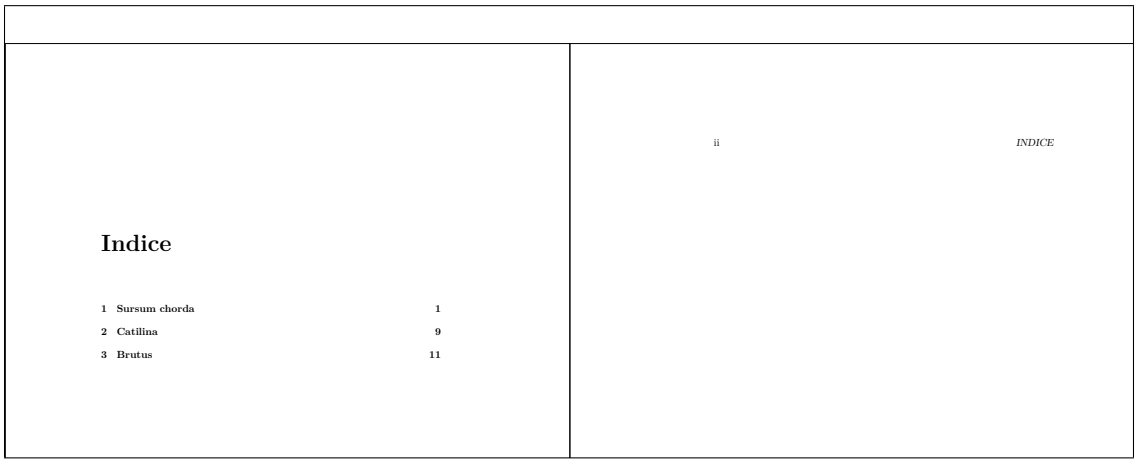


Figura 7: L'indice, ottenuto con un comando interno corrispondente ad un capitolo asteriscato, ma che sempre da macro interne ottiene il titolo e le testatine uguali anche sulle pagine dispari (non mostrate)

B Ridefinizione della macro `\tableofcontents`

Il comando `\tableofcontents` può venire ridefinito così:

```
\renewcommand\tableofcontents{%
    \if@twocolumn
      \@restonecoltrue\onecolumn
    \else
      \@restonecolfalse
    \fi
    \chapter*{\contentsname}
    \@starttoc{toc}%
    \if@restonecol\twocolumn\fi
}
```

Come si vede, a parte i soliti test per verificare se si sta componendo su due colonne, e a parte immettere il file `toc` del documento, il cuore della macro è ridotto semplicemente a

```
\chapter*{\contentsname}
```

perché il nuovo comando provvede da solo a configurare le testatine che, invece, nella definizione originale devono essere caricate espressamente.

Queste poche righe possono essere messe al posto dei commenti indicati nel codice contenuto nell'appendice A. Questa ridefinizione va bene per sostituire la definizione originale nella classe book usata per il collaudo; se si usasse la classe memoir non sarebbe necessaria, anche perché quella classe provvede in modo proprio a gestire le testatine in generale; per l'indice usa un particolare stile delle pagine che già contiene le testatine corrette.

Riferimenti bibliografici

Beccari, C. (2018a). *Il LaTeX Reference Manual commentato*. GuIt. URL http://www.guitex.org/home/images/doc/GuideGuIT/latexhandbookcommentato.pdf.

— (2018b). *Il pacchetto TOPtesi*. TUG. Leggibile con `texdoc toptesi-it` o, in inglese, con `texdoc toptesi`.

Beccari, C., Giacomelli, R. e Molinaro, M. (2018). «Il concorso della scacchiera». *ArsTEXnica*, (25).

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

Lamport, L. (1985). *LaTeX. A Document Preparation System*. Addison-Wesley, 1ª edizione.

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

The LaTeX Project Team (2018). *The xparse package*. TUG. Leggibile con `texdoc xparse` con una distribuzione TeX Live.

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

Esperienze nella pubblicazione di un volume di atti di convegno

Massimiliano Dominici

Sommario

L'articolo espone le esperienze fatte dall'autore durante la redazione e composizione tipografica di un volume di atti di convegno. Vengono discusse alcune delle problematiche che si possono incontrare nel corso di un lavoro di questo tipo e le soluzioni che sono state adottate per la gestione, in particolar modo, di una bibliografia voluminosa e di un contesto multilinguistico in cui si trovano a coesistere diversi alfabeti. Si avverte il lettore che alcune di queste soluzioni sono superate dallo sviluppo dei programmi usati.

Abstract

The article shows the experiences made by the author during the editing and typesetting of a volume of conference proceedings. Some issues arising in a job of this kind are discussed together with the solutions that have been adopted for the management, in particular, of a large bibliography and a multilingual context in which several alphabets coexist. The reader is warned that some of these solutions have been made obsolete by the development of the software used.

1 Introduzione

Nell'estate del 2012 venni contattato dal professor Pier Daniele Napolitani, con il quale già collaboravo all'Edizione nazionale dell'Opera Matematica di Francesco Maurolico, per sondare la mia disponibilità a occuparmi dell'impaginazione di un volume di atti di un convegno che si sarebbe svolto di lì a poco in memoria di Pierre Souffrin,[1] studioso di storia della scienza scomparso dieci anni prima: *Science et Représentations. Colloque International en mémoire de Pierre Souffrin* (Souffrin). Il lavoro vero e proprio sarebbe iniziato circa un anno e mezzo dopo e si sarebbe protratto per un biennio, prima della pubblicazione (Caye *et al.*, 2015).

I motivi che avevano indotto il professor Napolitani a rivolgersi a me (e quindi a usare LaTeX per la composizione del volume) erano diversi. Innanzi tutto la familiarità con lo strumento e la consapevolezza della qualità finale del risultato. Poi la versatilità del programma che consentiva, eventualmente, di cambiare in corsa alcune decisioni riguardanti l'impaginazione o l'aspetto del volume senza troppe conseguenze. Non ultimo il fatto che, lavorando a stretto contatto, era possibile unificare le fasi di redazione e impaginazione, velocizzando così il lavoro.

Come si vedrà nel corso dell'esposizione, l'impiego di LaTeX è stato essenziale a garantire che tutte le fasi si svolgessero in relativa tranquillità, sperimentando anche, soprattutto per la bibliografia, diverse soluzioni man mano che si evidenziavano nuovi problemi e venivano alla luce nuove esigenze.

È stato un lavoro sicuramente molto impegnativo, in cui alla fine LaTeX è stato usato in tutta la sua ampiezza come strumento di pubblicazione avanzato, in un contesto multilinguistico e in presenza di soluzioni di impaginazione ad ampio spettro. Paradossalmente è stata invece messa alla prova in misura minore la sua capacità di comporre formule matematiche. Le poche presenti nel volume non presentavano infatti particolari difficoltà.

Questo mi ha consentito di acquisire un bagaglio di esperienze, nel bene e nel male, che cercherò di condividere con i lettori: quelle positive per fornire spunti e quelle negative come avvertimento a non ripetere i miei errori.

Un avvertimento importante che devo premettere all'articolo è che una parte significativa del codice effettivamente scritto all'epoca è (purtroppo) già obsoleto, in particolare quello relativo alla personalizzazione della bibliografia; quanto segue, quindi, si concentrerà più sulle procedure e sui concetti piuttosto che sull'illustrazione di frammenti di codice da usare "in produzione".

2 Caratteristiche del lavoro e specifiche dell'editore

Il volume avrebbe dovuto raccogliere una trentina di contributi[2] scritti in italiano, in inglese o in francese, ciascuno corredato della propria bibliografia. Alcuni contributi (pochi) sarebbero stati consegnati in LaTeX, la maggior parte in qualche versione del formato di Word (da RTF a DOCX). Quasi tutti i contributi contenevano delle illustrazioni realizzate in vari formati, per lo più raster.

Oltre alle tre lingue principali erano previste singole parole o brevi passi in latino, greco, tedesco, arabo, arabo traslitterato e persiano traslitterato.

1. Per un ricordo della figura umana e di studioso di Pierre Souffrin il lettore può consultare Napolitani e Gautero (2006).

2. Alla fine sono stati ventisei, a fronte dei trentadue interventi al convegno.

Ogni autore avrebbe dovuto ricevere, una volta completata, la bozza impaginata del proprio articolo con la relativa bibliografia, in modo da poterla esaminare e trasmettere alla redazione le eventuali correzioni.

Per gli autori che avrebbero usato Word erano state preparate alcune semplici raccomandazioni in cui si chiedeva, fondamentalmente, di non usare citazioni bibliografiche estese nel corpo del testo o in nota ma di usare un'etichetta identificativa del tipo (Autore:anno). Le voci bibliografiche avrebbero dovuto essere riportate alla fine del contributo, separate l'una dall'altra da una riga vuota e precedute, ciascuna, dall'etichetta con la quale era richiamata nel testo. L'intenzione era quella di evitare una pletora di brevi note a contenuto bibliografico e, allo stesso tempo, di poter trasformare velocemente l'etichetta in un comando `\cite` funzionante per mezzo di un semplice *cerca e sostituisci*. Nel volume, poi, la bibliografia sarebbe stata unificata a fine volume e non separata per contributi, in modo da evitare duplicazioni e rendere coerente l'insieme delle singole voci.

2.1 Organizzazione delle cartelle e del codice

Per convenienza i contributi degli autori sono stati separati dalle parti "accessorie" (frontespizio, prefazione, introduzione, bibliografia, indice). Ogni contributo è stato "isolato" in una specifica cartella insieme alle immagini da includere. Nella figura 1 è riportata sommariamente la struttura. Si noterà che, oltre al file `atti_vinci.tex` che rappresenta il file *master*, al primo livello della struttura si trovano tre file `.sty`. Questi file contengono il codice del preambolo, che nel corso del tempo è diventato davvero troppo voluminoso perché potesse essere gestito facilmente come un tutto unico (soprattutto quello relativo alla bibliografia): di qui l'esigenza di suddividerlo in moduli per una più agevole manutenzione.

Compare anche un file `template.tex` di cui si parlerà nel paragrafo 3.3.

2.2 Specifiche dell'editore

Per quanto riguarda le specifiche, l'editore ha fornito indicazioni sul carattere da usare (Simoncini Garamond per il testo e New Aster per gli esponenti di nota); i corpi principali (testo 11/12, infratesto 10/11, note 8.5/8.5); il formato finito (17×24 cm) e le dimensioni della gabbia. Per i dati mancanti, invece, si rimandava alla consultazione di altri volumi della stessa collana. In alcuni casi questo lavoro di estrapolazione non si è rivelato del tutto agevole: per esempio, solo dopo alcuni scambi con la redazione della casa editrice abbiamo scoperto che il filetto di separazione delle note non andava posizionato a una distanza fissa dal testo ma a metà tra il blocco del testo e quello delle note. D'altra parte la casa editrice ci ha lasciato una

```
Atti
├── Bibliografia
│   └── bibliografia.bib
├── Bozze_singoli_contributi
│   ├── Autore1
│   │   ├── Autore1_Fig1.jpeg
│   │   └── Contributo1.tex
│   └── Autore2
│       └── Contributo2.tex
├── Frontespizio
│   └── Frontespzizio.tex
├── Indice
│   └── Indice_nomi.tex
├── Introduzione
│   ├── Presentazione.tex
│   ├── Introduzione.tex
│   └── Introduzione_eng.tex
├── atti_vinci.tex
├── av_bibliografia.sty
├── av_layout.sty
├── av_lingue.sty
└── template.tex
```

Figura 1: Layout delle cartelle del progetto.

certa flessibilità nel definire l'aspetto di altri elementi come, per esempio, i titoli di paragrafo. Ci è stata garantita ampia libertà anche per quanto riguarda la scelta dei caratteri da usare nei contesti linguistici per i quali il Simoncini Garamond non fornisce adeguato supporto (si veda, per dettagli, il paragrafo 4.3).

3 La cassetta degli attrezzi

Come per qualsiasi lavoro, una cassetta di attrezzi funzionali è il prerequisito per trovarsi a proprio agio e procedere speditamente. Fermo restando che consuetudine e preferenze personali sono uno degli elementi fondamentali per la scelta di uno strumento piuttosto che un altro, va anche detto che non tutti gli "attrezzi" possono garantire la stessa flessibilità d'uso, né l'aderenza alle necessità del caso concreto. Di seguito elencherò i programmi usati per il progetto oggetto dell'articolo, evidenziando i pregi e le criticità emersi nel corso del lavoro.

3.1 Sistema operativo e distribuzione TEX

La scelta del sistema operativo è stata immediata, dal momento che sia io sia il curatore eravamo abituati a lavorare su Linux. Ai fini del progetto la specifica distribuzione non era rilevante. Uno dei vantaggi offerti da Linux (o da sistemi Unix in generale) è la facilità con cui è possibile realizzare semplici script che automatizzino alcune parti del lavoro. Mostrerò un paio di esempi in seguito (3.2).

La distribuzione TEX usata è stata TEX Live 2014 nella versione completa distribuita da TUG.

La versione di TEX Live è cambiata mentre il progetto era ancora in corso ma abbiamo deciso di continuare a usare la 2014 per evitare eventuali problemi di compatibilità. Questa decisione non mi ha impedito di installare in parallelo anche TEX Live 2015 e di usare l'una o l'altra secondo il bisogno. Istruzioni particolareggiate su come installare e gestire distribuzioni multiple di TEX Live su Linux si possono trovare nell'articolo di Enrico Gregorio scritto per ArsTEXnica 10 (Gregorio, 2010).

3.2 Controllo di versione e condivisione dei file

Considerando che i file avrebbero dovuto essere condivisi con persone il cui grado di familiarità con la tecnologia poteva essere molto variabile, abbiamo deciso di servirci di Dropbox per la sua facilità d'uso. La cartella condivisa di Dropbox sarebbe servita sostanzialmente per me, per il professor Napolitani e gli altri curatori e per la redazione della casa editrice. Lo scambio delle bozze con gli autori sarebbe invece avvenuto per email come da tradizione. Il sistema era stato pensato per ridurre al minimo gli attriti. Alla fine, ciò che conta è la comodità degli autori e non quella dei redattori.

Naturalmente io usavo un controllo di versione locale, sotto forma di repository `git` (Git), dove avveniva il lavoro reale, evitando così di modificare direttamente i file della cartella condivisa di Dropbox. Uno dei vantaggi di `git` è quello di poter ottenere facilmente un archivio dei soli file sotto controllo di versione, che in seguito può essere scompattato nella cartella condivisa. Questo è un classico compito da automatizzare con uno script della shell. Il file in questione, chiamato con molta fantasia `to-dropbox.sh`, conteneva le seguenti righe di codice:

```
#!/bin/bash

git archive \
  --output=../Servizio/atti.tar.gz master
tar -xvf ../Servizio/atti.tar.gz \
  -C '~/Dropbox/Seconde Bozze Vinci/'
```

3.3 Editor di testo

Un buon editor di testo, possibilmente integrato con la distribuzione LATEX, è uno strumento essenziale per chi deve modificare un testo e compilarlo. La possibilità di scelta è piuttosto ampia e va da editor dedicati (tra cui TeXworks può essere considerato quello "ufficiale") a plugin apposite per editor "generalisti".

Anche nel caso dell'editor di testo io e il curatore condividevamo le stesse preferenze, così è stato naturale scegliere Emacs, integrato con AUCTEX che è il plugin consigliato per LATEX. AUCTEX offre un ambiente di lavoro molto avanzato che va dall'evidenziazione della sintassi all'uso di funzioni per inserire, in maniera intelligente, le macro di uso più comune; dalla possibilità di ricercare e inserire riferimenti interni o citazioni bibliografiche alla compilazione parziale o totale del documento. Il Gruppo Utilizzatori Italiani di TEX offre tra la sua documentazione un'introduzione a AUCTEX (de Bari, 2006) e una guida più approfondita (Giordano *et al.*, 2013), alle quali si può fare riferimento per una panoramica completa.

Tuttavia, oltre all'ottimo supporto per LATEX fornito da AUCTEX e alle funzioni tipiche di un editor di testo tradizionale, Emacs, grazie al fatto di mettere a disposizione dell'utente un interprete Lisp, non solo è in grado di modificare e personalizzare le proprie funzioni interne, ma anche di interagire e integrarsi con l'ambiente circostante.

Questo può essere comodo per automatizzare alcuni aspetti del lavoro. Nel caso in esame, per evitare discrepanze nella compilazione dei singoli articoli, la compilazione avveniva a partire da un template che conteneva il preambolo comune (che man mano doveva essere modificato e adattato alle nuove circostanze, nonché corretto, quando ci si accorgeva di qualche errore) e includeva come file esterno il contributo su cui si concentrava il lavoro in quel momento. Quindi la sequenza di azioni da intraprendere comprendeva: aprire il template, modificarlo in modo che il file incluso fosse quello voluto, infine aprire il contributo e cominciare il lavoro. Sarebbe stato più comodo aprire direttamente il file del contributo e lasciare che di tutto il resto si occupasse il sistema. Emacs permette di fare una cosa del genere. Probabilmente è possibile farlo interamente con funzioni Elisp;[3] nel mio caso ho preferito una combinazione di funzioni Elisp e script di shell.

Lo script bash è visibile nella figura 2a e, come si può vedere, impiega `sed` per modificare il template sostituendo, nel file passato come terzo argomento, il contenuto della "variabile" LATEX `\DIR` con il primo argomento e il contenuto di `\ART` con il secondo argomento, cioè rispettivamente il nome della directory e del file del contributo da compilare. Già che c'è, lo script dà anche una ripulita, cancellando i prodotti della precedente compilazione della bibliografia.

Questo script viene richiamato da una funzione Elisp: `md-modify-template` (nel codice della figura 2b, preceduta dall'impostazione di due variabili). Questa funzione potrebbe essere richiamata dall'interno di Emacs[4] tramite il suo interprete (associato alla scorciatoia `META + x`, nel gergo Emacs `M-x`, di solito equivalente a `ALT + x` su un PC Linux), ma possiamo fare di meglio, cioè eseguire automaticamente la procedura all'apertura del file.

Emacs ha un sistema per assegnare a specifiche variabili dei valori confinati a un particolare file (o

3. Elisp sta per *Emacs*Lisp, cioè il dialetto Lisp tipico di Emacs.

4. In realtà no, perché manca la direttiva `(interactive)`.

```
#! /bin/bash

rm -f ../../template.bcf \
 ../../template.run.xml \
 ../../template.bbl
sed -i[old] \
  -e "s/\\\\def\\\\DIR{.*}/\\\\def\\\\DIR{$1}/g" \
  -e "s/\\\\def\\\\ART{.*}/\\\\def\\\\ART{$2}/g" "$3"
```

(a) Codice dello script della shell.

```
(defvar md-template-path "../../template.tex"
  "Template path")
(defvar md-template-modify-cmd-string "../../modify-template.sh"
  "Default command to be executed")
(defun md-modify-template (directory file)
  "The function modifies the file included in the template. Not interactive, for use in file variables."
  (call-process-shell-command md-template-modify-cmd-string nil 0 nil directory file md-template-path))
```

(b) Codice delle funzioni Elisp.

```
%% Local Variables:
%% mode: latex
%% TeX-master: "../../template"
%% TeX-engine: xetex
%% eval: (md-modify-template (file-name-base (directory-file-name default-directory)) (file-name-base buffer-file-name))
%% bibtex-dialect: biblatex
%% End:
```

(c) Variabili d'ambiente.

Figura 2: Il codice usato per modificare automaticamente il template comune all'apertura del singolo contributo.

per essere precisi a un particolare buffer). Questo è utile, per esempio nel nostro caso, per impostare il tipo di compilatore (`pdftex`, `xetex`, `luatex`, ecc.), il formato (`tex`, `latex`, ecc.), il "dialetto" del compilatore bibliografico (`bibtex` o `biblatex`) e altre cose che hanno a che fare direttamente con LaTeX o con l'editor di testo in generale. Qualcosa di simile alle righe magiche usate da molti altri editor di testo (e che, per inciso, le precede cronologicamente). Trattandosi però di un interprete Lisp, non siamo limitati all'impostazione di una variabile, ma possiamo eseguire una funzione tramite la parola chiave `eval` (figura 2c).[5] Alla funzione vengono automaticamente passati il nome del file e della directory di lavoro, che sono noti a Emacs.

3.4 Altri programmi

Come anticipato nel paragrafo 2, la maggior parte dei contributi sono pervenuti sotto forma di documento Word o, più raramente, RTF o ODT. Quindi era necessario un programma in grado di aprirli e visualizzarli (eventualmente anche modificarli, ma questa funzionalità non era essenziale). La scelta è caduta su LibreOffice Writer, che si è rivelato perfettamente adeguato alle nostre esigenze.

Per la conversione dei file da DOC, DOCX, RTF o ODT a LaTeX (questo sì, essenziale!) ci siamo affidati a Writer2LaTeX (Just, 2018), un programma Java che può essere installato come plugin di LibreOffice Writer o usato in maniera autonoma da riga di comando. È in questa seconda modalità che è stato usato nel nostro progetto. L'output generato può essere personalizzato, entro certi limiti, tramite un file di configurazione. Il risultato finale è spesso ancora un po' troppo sporco perché cerca di tradurre in qualche modo gli stili applicati alle varie porzioni di testo da Word. Questi stili sono spesso spuri e derivano generalmente in maniera involontaria da copia-e-incolla. Il file risultante va quindi ripulito con pazienza, usando le funzioni di ricerca e sostituzione dell'editor di testo (nel nostro caso, Emacs). La trafila può essere un po' laboriosa, perché è difficile conoscere in anticipo il nome dello stile e quindi preparare in precedenza le funzioni di sostituzione e applicarle in batch. Inoltre Writer2LaTeX accetta in input solo file ODT, quindi i file di Word vanno precedentemente salvati in formato ODT, ma questo si può fare facilmente con LibreOffice.[6] All'epoca non esisteva ancora, ma un'alternativa valida a Writer2LaTeX oggi potrebbe essere `doc2tex`,[7] che permette di configurare l'output tramite semplici file `csv` o file XML se si vuole un controllo più fine.

5. Per ragioni di sicurezza, Emacs non esegue immediatamente il codice ma chiede se si vuole davvero eseguirlo e se la scelta debba essere registrata, in modo che le volte successive *questo* codice venga eseguito immediatamente, senza avvisi.

6. Anche da riga di comando, grazie al programma `unoconv`: http://dag.wiee.rs/home-made/unoconv/.

7. https://github.com/transpect/docx2tex.

Infine, nei casi in cui è stato necessario ritoccare delle immagini raster, è stato usato l'ottimo Gimp, che è il programma di elaborazione di immagini di riferimento per Linux. Per le immagini vettoriali, invece, è stato usato Ti*k*Z per le ragioni discusse nel paragrafo 4.5.1.

4 Il codice LaTeX

Come accennato già in precedenza, il codice contenente le varie impostazioni può essere suddiviso grossolanamente in tre parti, corrispondenti ai tre file `.sty` visibili nella figura 1. Altre personalizzazioni, non direttamente collegate all'impaginazione vera e propria, alla gestione dei contesti linguistici o alla bibliografia, sono state lasciate nel file "master".

Nei seguenti paragrafi verranno mostrati solo frammenti di codice quando ritenuto di particolare interesse o per ragioni esemplificative, mentre verranno discusse con più dettaglio le scelte fatte e le motivazioni che stanno alla base di tali scelte.

4.1 Compilatore

Prima di parlare delle singole impostazioni, va specificato che come compilatore è stato scelto XeLaTeX. Dato il carattere del volume, in cui era necessario passare spesso da un contesto linguistico a un altro, usando anche alfabeti non latini, e usare font proprietari e in ogni caso non distribuiti con TeX Live, era praticamente obbligatorio il ricorso a un compilatore che supportasse nativamente Unicode e OpenType: XeLaTeX o LuaTeX. La scelta è caduta sul primo perché all'epoca LuaTeX non aveva la stessa maturità mentre XeLaTeX garantiva una maggiore velocità di compilazione e questo aveva il suo effetto su un hardware non propriamente performante come quello che avevo a disposizione, specie per la compilazione del volume intero. Oggi probabilmente prenderei seriamente in considerazione l'idea di usare il secondo, dato che quasi tutte le limitazioni elencate sono venute meno.

4.2 Impostazioni tipografiche

In questa prima sezione, contenuta nel file `av_layout.sty`, sono state implementate le specifiche fornite esplicitamente o implicitamente dall'editore.

Dovendo modificare pesantemente le impostazioni di default di un documento LaTeX, ho deciso di usare memoir (Wilson e Madsen, 2018) come classe perché consente all'utente di rimodellare in maniera abbastanza semplice quasi tutti gli elementi tipografici del documento. Per i dettagli delle varie soluzioni si rimanda, anche dove non è detto esplicitamente, alla lettura del manuale.

4.2.1 Caratteri e dimensione dei corpi

Dal momento che il documento usa dei caratteri OpenType è bene usare il pacchetto fontspec (Robertson, 2018). Per le poche formule presenti non si rendeva necessario un supporto particolarmente avanzato, e dato che l'uso del Computer Modern non era un'opzione praticabile date le specifiche dell'editore, è stato impiegato mathspec (Gilbert Moschou, 2016) per prendere dai font del testo lettere e numeri da usare in contesto matematico. Sono inoltre necessari alcuni aggiustamenti relativi agli spazi tra elementi delle formule e i caratteri resi attivi da polyglossia vanno disattivati.

Come notato nel paragrafo 2.2, per gli esponenti delle note l'editore ha imposto un carattere diverso da quello del corpo del testo. In generale, il *Simoncini Garamond Std* della Linotype presenta diverse lacune, tra cui la mancanza di molti caratteri con diacritici indispensabili per la corretta resa di lingue come il polacco o la traslitterazione dell'arabo. Nei casi in cui un particolare carattere mancava si è dovuto sostituirlo con l'equivalente preso da un font più completo e simile al Simoncini; la scelta è caduta su *EB Garamond*, contenuto in TeX Live. Per fare questo è stata sfruttata una particolare caratteristica di XeLaTeX. In XeLaTeX è possibile assegnare un determinato carattere a una "classe" (di default i caratteri appartengono in genere alla classe 0 o 255) e poi eseguire del codice specifico al passaggio da una determinata classe a un'altra. Nella figura 3 si vede un esempio di sostituzione di caratteri nel nome proprio arabo persiano. Le lettere con diacritici sono in *EB Garamond*, le altre in *Simoncini Garamond*: la differenza è visibile, soprattutto a schermo.

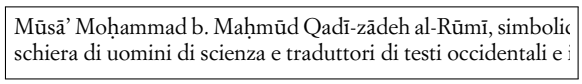


Figura 3: Sostituzione di caratteri nell'arabo traslitterato.

Il codice di caricamento dei caratteri principali del testo, quindi, è il seguente (fontspec è caricato implictamente da mathspec):

```
\usepackage{mathspec}
\setmainfont[Ligatures=TeX]{Simoncini Garamond Std}
\newfontfamily\aster{NewAster LT}
\setmathsfont(Digits,Latin){Simoncini Garamond Std}
\setmathsfont(Greek)[Scale=.895]{Old Standard}
\setminwhitespace[2000]
\everymath{\noitalian@shorthands}

\everydisplay{\noitalian@shorthands}
\newfontfamily\SubstFont{EB Garamond}[Scale=1.1]
\XeTeXinterchartokenstate=1
\newXeTeXintercharclass\Subst
\XeTeXcharclass"0100=\Subst % Ā
\XeTeXcharclass"0101=\Subst % ā
\XeTeXcharclass"0119=\Subst % ę
...
\XeTeXinterchartoks 0 \Subst = {\begingroup
                                \SubstFont}
\XeTeXinterchartoks 255 \Subst = {\begingroup
                                  \SubstFont}
\XeTeXinterchartoks \Subst 0 = {\endgroup}
\XeTeXinterchartoks \Subst 255 = {\endgroup}
```

L'elenco dei caratteri da sostituire sarebbe più lungo, ma ne ho mostrati solo un paio come esempio.

Tra i pochi elementi che non possono essere direttamente impostati tramite macro specifiche di memoir c'è la dimensione dei vari corpi: `\normalsize`, `\small`, ecc. vanno ridefiniti "a mano", come si vede nel codice seguente per il solo `\normalsize`. Si noti che la dimensione è specificata in punti postscript (*big points*, `bp`) perché questo è lo standard tipografico attuale[8] e che subito dopo la ridefinizione il corpo principale è "messo in funzione" tramite l'istruzione `\normalsize`, in modo che ogni ulteriore impostazione derivata da parametri della dimensione del corpo del testo (per esempio una misura specificata in `em`) tenga in considerazione la modifica in questione.

```
\renewcommand\normalsize{%
  \@setfontsize\normalsize{11bp}{12bp}%
  \abovedisplayskip 11bp \@plus3bp \@minus6bp
  \abovedisplayshortskip \z@ \@plus3bp
  \belowdisplayshortskip6.5bp\@plus3.5bp\@minus3bp
  \belowdisplayskip \abovedisplayskip
  \let\@listi\@listI}
\normalsize
```

4.2.2 Impostazioni della pagina

La classe memoir permette di impostare abbastanza agevolmente e in modo intuitivo le varie dimensioni relative al formato della pagina e della gabbia del testo. In genere, le varie macro permettono di specificare dimensioni assolute e/o rapporti tra dimensioni. Le macro sono fatte in modo tale che non è necessario riempirne tutti gli argomenti: quelli superflui possono essere tralasciati, specificando al loro posto il carattere `*`. Per fare un esempio, la macro `\settrimmedsize`, che serve a impostare le dimensioni del formato finale della pagina, prende come argomenti l'altezza, la larghezza e il loro rapporto. Ovviamente una di queste informazioni è superflua, perché una volta fissate le prime due la terza è determinata di conseguenza. Se si specificano altezza e larghezza, quindi, come nel codice che segue, specificare anche il rapporto è inutile,[9] per cui quest'ultimo può essere indicato semplicemente con `*`.

```
\setstocksize{240mm}{170mm}
\settrimmedsize{240mm}{170mm}{*}
\settypeblocksize{187.8mm}{122.8mm}{*}
\setlrmargins{22mm}{*}{*}
\setulmargins{71.4pt}{*}{*}
\setheaderspaces{*}{10pt}{*}
\checkandfixthelayout[nearest]
\parindent=6mm
```

8. Alcune case editrici, però, continuano a usare come riferimento il punto tipografico continentale o *didot*, in gergo TEX `dd`.

9. Può perfino essere fonte di confusione, se viene specificato un rapporto sbagliato. In questi casi memoir ha delle regole predefinite per calcolare le dimensioni, ma è sconsigliato comunque inserire specifiche superflue. Il manuale, al quale si rimanda per ulteriori dettagli, spiega quali sono queste regole.

Si noti che per le macro `\setlrmargins` e `\setulmargins` il numero corretto di parametri da specificare è uno, perché assumono che sia già stata specificata la dimensione dell'area del testo tramite `\settypeblocksize`, per cui è sufficiente specificare la dimensione di un solo margine o il solo rapporto tra i margini. In quei casi in cui le specifiche dell'editore prevedono solo le dimensioni dei margini e non quelle dell'area del testo è più conveniente usare le macro `\setlrmarginsandblock` e `\setulmarginsandblock` e specificare due valori (i due margini, oppure un margine e il rapporto): l'area del testo ne risulterà determinata automaticamente.

Una volta impostati i valori, la macro `\checkandfixthelayout` esegue i calcoli necessari e restituisce la geometria della pagina. Poiché di norma l'altezza impostata non sarà un multiplo intero dell'avanzamento di riga, bensì cadrà nell'intervallo tra due di questi multipli, `\checkandfixthelayout` si incarica anche di aggiustare questo parametro. L'algoritmo in base a cui opera l'aggiustamento può essere facoltativamente scelto dall'utente. Nel nostro caso abbiamo indicato che aggiusti al multiplo più prossimo, sia esso quello superiore o quello inferiore.

Infine vanno impostati alcuni parametri che impediscano l'eccessivo saltellamento delle righe, in presenza di maiuscole accentate per esempio, e l'inserimento di spazi bianchi tra i capoversi. Sono entrambi considerati "orrori tipografici" dalle case editrici di tradizione umanistica.

```
\parskip=0pt
\newlength\oldlineskiplimit
\oldlineskiplimit=\lineskiplimit
\lineskiplimit=-\maxdimen
```

Prima di reimpostare `\lineskiplimit` salviamo il suo valore di default, perché potranno capitare situazioni di emergenza in cui sarà necessario ripristinarlo.

Azzerare il valore di *stretch* di `\parskip` significa togliere flessibilità al sistema e ritrovarsi con qualche vedova o orfana in più o, peggio ancora, in situazioni (specie in presenza di note a piè di pagina) in cui un'impaginazione decente non è possibile. In questi casi è necessario intervenire manualmente allungando o accorciando qualche capoverso con `\looseness`. Non sempre questo è sufficiente a risolvere il problema: nel caso del nostro progetto, il curatore aveva fortunatamente un margine di intervento sul testo sufficientemente ampio da poter riformulare qualche espressione in vista di un guadagno dal punto di vista tipografico.

4.2.3 Titoli e titoletti

Il volume è diviso in quattro sezioni, contenenti ciascuna diversi contributi. Da un punto di vista strutturale, quindi, è stato immediato assimiliare queste divisioni maggiori a "parti" e i singoli

contributi a "capitoli", a loro volta suddivisi in paragrafi e sottoparagrafi.

L'unica nota di rilievo riguarda il modo di trattare gli autori e il titolo del contributo. Similmente a quanto avviene per il titolo e gli autori di un documento, sono state introdotte le macro `\AVtitle` e `\AVauthor` che assegnano a una variabile interna il testo del titolo e rispettivamente degli autori, e una macro `\AVmaketitle` che, richiamando internamente la macro `\chapter`, stampa in seguito il titolo del contributo correttamente composto. È possibile specificare, sia per il titolo che per gli autori, un testo differente da usare nelle testatine e nell'indice.

Per quanto riguarda le macro per impostare lo stile dei capitoli, dei paragrafi e delle testatine, rimando al manuale di memoir, notando qui soltanto che è possibile definire più di uno stile da applicare in parti differenti del documento. Nel nostro caso questo è stato utile per impostare, in alcune specifiche porzioni di un contributo, dei titoli di paragrafo centrati invece che imbandierati a sinistra come nel resto del volume.

4.3 Supporto multilinguistico

Il secondo grosso blocco di impostazioni ha riguardato il supporto multilinguistico. Il pacchetto usato per specificare le varie lingue del documento e i font associati a ciascuna è stato polyglossia (Charette e Reutenauer, 2018). Come anticipato nel paragrafo 2, oltre alla lingua principale, l'italiano, e le altre due lingue in cui poteva essere presentato un contributo (inglese o francese), erano presenti porzioni più o meno lunghe di testo in tedesco, latino, arabo o persiano traslitterato, arabo e greco, nella variante politonica. Per questi due ultimi contesti linguistici è stato necessario ricorrere a font specifici:

```
\newfontfamily\greekfont[Scale=.895]{Old Standard}
\newfontfamily\arabicfont[Script=Arabic]{Amiri}
```

Per quanto riguarda il greco c'è un'ulteriore particolarità. Nella maggior parte dei contributi il testo greco era correttamente codificato Unicode. In due contributi, però, era stato inserito con codifiche pre-Unicode, che presuppongono font specifici, e che, dal punto di vista del convertitore da DOC a LaTeX, sono testo ASCII a tutti gli effetti. Fortunatamente è bastato fare qualche piccolo adattamento perché i testi in questione fossero nella tradizionale codifica LaTeX per il greco, basata su ASCII. A questo punto mancava di farla riconoscere a XeTeX che di norma accetta solo un input in codifica Unicode. Per questo scopo si può usare un file che "mappi" i caratteri (o gruppi di caratteri) ASCII nei rispettivi equivalenti Unicode e farlo leggere al compilatore. Si parte da un file `asciitogreek.map` che dovrà essere compilato con il programma `teckit_compile`, ottenendo `asciitogreek.tec`. Una volta posto questo file nella cartella di lavoro, si definiranno nel preambolo degli ambienti e dei comandi per marcare il testo greco in codifica ASCII:

```
\XeTeXinputnormalization=1

\newenvironment{transgreek}
  {\catcode`\~=12 \catcode`\*=12
   \begin{otherlanguage}{greek}%
   \addfontfeature{Mapping=asciitogreek}}
  {\end{otherlanguage}}
\newenvironment{transgreek*}
  {\catcode`\~=12 \catcode`\*=12
   \begin{otherlanguage*}{greek}%
   \addfontfeature{Mapping=asciitogreek}}
  {\end{otherlanguage*}}

\newcommand{\texttransgreek}{\begingroup
  \catcode`\~=12 \catcode`\*=12
  \innertexttransgreek}
\newcommand*{\innertexttransgreek}[1]{%
  \begin{transgreek*}#1\end{transgreek*}\endgroup}
```

Il file `asciitogreek.map`, di cui è autore Enrico Gregorio, si può trovare qui: http://profs.sci.univr.it/~gregorio/asciitogreek.map. Il suo uso, comprese le macro riportate sopra, è illustrato in una discussione del forum del Gruppo Utilizzatori Italiani di TeX: https://www.guitex.org/home/en/forum/5-tex-e-latex/56523.

4.4 Bibliografia

La parte più corposa di modifiche riguarda l'aspetto della bibliografia. Il pacchetto biblatex (Lehmann, 2018) fornisce la possibilità di personalizzare a piacere il formato sia delle voci bibliografiche raccolte a fine volume, sia delle citazioni nel testo. È stato deciso di usare un formato di citazione molto semplice, nella forma "Autore anno", con la possibilità di usare, specialmente per collezioni di opere, un'abbreviazione di uso comune nella letteratura specialistica. Per esempio, le opere complete di Keplero (*Johannes Kepler gesammelte Werke*) sono state identificate con l'etichetta "JKGW".

Più complicato il formato della bibliografia finale dove, dopo vari tentativi, ci si è orientati su un compromesso tra una voce bibliografica ricalcata sugli standard delle pubblicazioni umanistiche e il richiamo dell'etichetta per le citazioni. Il risultato finale può essere visto nella figura 4, il cui primo frammento riporta anche i criteri a cui ci si è attenuti nella compilazione delle voci. Si noterà che ogni voce completa è preceduta dall'etichetta usata nelle citazioni e dal segno "=". Nel riquadro in basso, alcune delle opere di Keplero riportano un riferimento incrociato all'edizione delle opere complete. Nel nostro progetto si è scelto di usare per i riferimenti interni il campo `xref` al posto del più tradizionale campo `crossref`, in modo da indicare solo l'etichetta della voce richiamata e non l'intero contenuto.

biblatex offre molti strumenti per la personalizzazione della bibliografia. Molti sono accessibili tramite opzioni del pacchetto, altri usando macro

BIBLIOGRAFIA

Sono riuniti in questa sezione del volume tutti i riferimenti bibliografici presenti nei vari articoli che lo compongono; trattandosi di un un volume in più lingue (italiano, francese, inglese) abbiamo dovuto necessariamente operare delle scelte. In particolare le abbreviazioni bibliografiche sono tutte in italiano (ARISTOTELE e non ARISTOTELES; EGIDIO ROMANO e non GILLES DE ROME).

Le singole voci seguono però le caratteristiche dell'opera citata: per esempio nei titoli di opere inglesi le iniziali di nomi e aggettivi sono maiuscole; nelle opere tedesche per indicare le pagine si usa S. e non pp.; nei testi francesi «édité par» e non «a cura di» e così via.

Per le opere più antiche e in particolare per quelle latine si è scelto di riportare stralci del frontespizio che contengono il nome dell'autore.

AF 1998 = *La grande storia dell'artigianato (Arti fiorentine). Volume primo, Il Medioevo*, Firenze, Cassa di Risparmio di Firenze, Giunti, 1998.

AGRICOLA 1546 = *Georgii Agricolae De ortu et causis subterraneorum, Lib. V. De natura eorum quae effluunt ex terra, Lib. IIII. De Natura fossilium, Lib. X. De veteribus et novis metallis, Lib. II. Bermannus, sive De re metallica Dialogus. Interpretatio Germanica vocum rei metallicae, addito indice foecundissimo*, Basel, H. Froben & N. Episcopius, 1546.

CLAGETT 1964-1984 = Marshall Clagett, *Archimedes in the Middle Ages*, 1964-1984; I, *The Arabo-Latin Tradition*, Madison, The University of Wisconsin Press, 1964; II, *The Translations from the Greek by William of Moerbeke*, Philadelphia, The American Philosophical Society, 1976; III, *The Fate of the Medieval Archimedes, 1300 to 1565*, ibid., 1978; IV, *A Supplement on the Medieval Latin Traditions of Conic Sections (1150-1156)*, ibid., 1980; V, *Quasi-Archimedean Geometry in the Thirteenth Century*, ibid., 1984.

– 1968 = Marshall Clagett, *Nicole Oresme and the Medieval Geometry of Qualities and Motions. A Treatise on the Uniformity and Difformity of Intensities Known as Tractatus de configurationibus qualitatum et motuum*, Madison, Milwaukee, & London, The University of Wisconsin Press, 1968.

KEPLER 1596 = *Prodromus Dissertationum Cosmographicarum, Continens Mysterium Cosmographicum … a Iohanne Keplero*, Tübingen, Gruppenbach, 1596 (traduzione francese in KEPLER 1984b).

– 1604a = *Paralipomena ad Vitellionem seu astronomia optica … authore Iohanne Keplero*, Frankfurt am Main, C. de Marne & J. Aubry, 1604.

– 1604b = Johannes Kepler, *Ad Vitellionem paralipomena, quibus astronomiae pars optica traditur*, in JKGW, vol. II, pp. 5–391.

– 1606 = *Iohannis Kepleri … Stella nova in pede Serpentarii, at qui sub ejus exortum de novo iniit, Trigono Igneo. Libellus Astronomicis, Physicis, Metaphysicis … Disputationibus … plenus. Accesserunt I. De stella incognita Cygni: Narratio Astronomica II. De Jesu Christi … Vero Anno Natalitio …*, Praha & Frankfurt am Main, P. Sesse & W. Richter, 1606.

– 1608 = *Aussführlicher Bericht von dem newlich im Monat Septembri und Octobri diß 1607 Jahrs erschienen Haarstern oder Cometen und seinen Bedeutungen … Gestellet durch Joannem Kepplern*, 1608, in KEPLER 1859-1871, vol. VII.

– 1609a = Johannes Kepler, *Astronomia nova αἰτιολογητός, seu physica coelestis, tradita commentariis de motibus stellae Martis plurium annorum pertinaci studio elaborata Pragae a … Iohanne Keplero*, Heidelberg, G. Vögelin, 1609.

– 1609b = Johannes Kepler, *Astronomia nova αἰτιολογητός, seu physica coelestis, tradita commentariis de motibus stellae Martis*, in JKGW, vol. III, pp. 5–424.

Figura 4: Alcuni stralci dalla bibliografia del volume.

specifiche messe a disposizione degli autori. Per alcune cose, però, bisogna necessariamente ricorrere alla ridefinizione di macro. Qui di seguito offro alcuni esempi, quelli che ritengo più meritevoli di nota, anche se alcuni sono divenuti obsoleti nel corso degli anni.

Per iniziare, il file `av_bibliografia.sty` richiama il pacchetto biblatex con una serie di opzioni, come si vede nel codice che segue, che già da sé impostano a grandi linee l'aspetto della bibliografia nella maniera desiderata dall'autore.

```
\RequirePackage[bibstyle=authortitle-comp,
                citestyle=authoryear-comp,
                uniquename=false,
                uniquelist=false,
                sorting=nyt,
                maxnames=2,
                backend=biber,
                dashed=false,
                isbn=false,
                usetranslator=true,
                mincrossrefs=99,
                autolang=other,
                dateabbrev=false,
                eventdate=comp,
                pagetracker=page,
                ]{biblatex}
```

È da notare che viene fatta una scelta un po' inconsueta, cioè lo stile di citazione è basato su quello "autore-anno", mentre lo stile delle voci bibliografiche è derivato da quello "autore-titolo". Questo per ottenere il compromesso di cui si parlava sopra tra standard delle pubblicazioni umanistiche in bibliografia e semplicità nelle citazioni. Viene inoltre impostata la soglia del numero di autori (o curatori) oltre il quale scatta l'abbreviazione automatica (nome del primo autore et al.); viene di fatto disabilitata l'aggiunta automatica di una voce richiamata da altre voci (per esempio un volume di atti richiamato dalla voce bibliografica dei singoli contributi) anche se non citata esplicitamente; e si impone che ogni voce bibliografica sia correttamente impostata secondo la sua lingua originale tramite l'ambiente otherlanguage.

Alcune personalizzazioni veloci possono essere fatte, per esempio, per cambiare, o aggiungere, parole o frasi dipendenti dal contesto linguistico. In concreto, nel nostro caso abbiamo cambiato il separatore di intervalli di pagina, che appartiene agli *extras* applicati ad ogni cambio di lingua, e alcune espressioni tipiche. Riporto solo il codice riguardante l'italiano ma il concetto è valido anche per le altre lingue:

```
\DefineBibliographyExtras{italian}{%
  \def\bibrangedash{-}
}
\DefineBibliographyStrings{italian}{%
  nodate = {in\space stampa},
  editor           = {a cura di},
  editors          = {a cura di},
}
```

Anche la ridefinizione del formato di un singolo "campo", per esempio il *titolo* di un'opera, è abbastanza semplice da ottenere. Con il codice seguente si fa in modo che per la maggior parte delle tipologie di opere il titolo sia in corsivo, per altre sia delimitato da virgolette basse.

```
\DeclareFieldFormat[article,inproceedings,inbook,
                    incollection,thesis,
                    suppcollection,
                    unpublished]{title}{\emph{#1}}
\DeclareFieldFormat[suppbook]{title}{«#1»}
\DeclareFieldFormat[periodical]{title}{«#1»}
```

Più complicato ridefinire una "suddivisione logica" della voce bibliografica di più alto livello. Sempre prendendo in considerazione il titolo di un'opera come esempio, biblatex deve infatti tener conto della eventuale presenza anche di un sottotitolo e adattare di conseguenza il formato. Per questo motivo un *driver*, cioè il codice che gestisce l'aspetto di una tipologia di voce bibliografica secondo un determinato stile, è organizzato in macro, che a loro volta richiamano al loro interno i campi che abbiamo visto sopra. Quindi, per fare in modo che un sottotitolo sia stampato in tondo invece che in corsivo, dobbiamo modificare la macro `title`:

```
\renewbibmacro*{title}{%
  \ifboolexpr{
    test {\iffieldundef{title}}
    and
    test {\iffieldundef{subtitle}}
  }
  {}
  {\printtext[title]{%
     \printfield[titlecase]{title}}%
   \setunit{\subtitlepunct}%
   \printfield[titlecase]{subtitle}
   \newunit%
  }%
  \printfield{titleaddon}%
}
```

La macro `\printfield` si limita a stampare il contenuto del campo, mentre `\printtext` applica il formato. Se avessimo voluto un formato speciale per i sottotitoli, avremmo potuto definirlo con la macro `\DeclareFieldFormat`. Nel nostro caso non era però necessario. A volte è stato necessario ridefinire un'intera macro, come nel codice precedente, solo per cambiare la punteggiatura.

Anche per riordinare i vari elementi all'interno delle voci bibliografiche o per aggiungerne alcuni è necessario ridefinire l'intero *driver*. Nella figura 5 è riportato il codice del *driver* per il tipo `article`, modificato in modo da aggiungere casa editrice e luogo di edizione, se disponibili.

Nella figura si può vedere che una delle prime macro che vengono richiamate dal *driver* è `begentry`. Insieme a `finentry` essa è presente in tutti i *driver* e serve a inserire eventuale codice da parte dell'autore (come impostazione predefinita è vuota). È possibile quindi sfruttare questa macro per inserire l'etichetta che apparirà poi nelle citazioni.

biblatex permette di manipolare i dati dei vari record bibliografici per ottenere effetti molto complicati. Per esempio, nel riquadro centrale della

```
\DeclareBibliographyDriver{article}{%
  \usebibmacro{bibindex}%
  \usebibmacro{begentry}%
  \usebibmacro{author/translator+others}%
  \setunit{\labelnamepunct}\newblock
  \usebibmacro{title}%
  \newunit
  \printlist{language}%
  \newunit\newblock
  \usebibmacro{byauthor}%
  \newunit\newblock
  \usebibmacro{bytranslator+others}%
  \newunit\newblock
  \printfield{version}%
  \newunit\newblock
  \usebibmacro{in:}%
  \usebibmacro{journal+issuetitle}%
  \newunit
  \usebibmacro{byeditor+others}%
  \newunit
  \usebibmacro{publisher+location}%
  \newunit
  \usebibmacro{note+pages}%
  \newunit\newblock
  \iftoggle{bbx:isbn}
    {\printfield{issn}}
    {}%
  \newunit\newblock
  \usebibmacro{doi+eprint+url}%
  \newunit\newblock
  \usebibmacro{addendum+pubstate}%
  \setunit{\bibpagerefpunct}\newblock
  \usebibmacro{pageref}%
  \newunit\newblock
  \iftoggle{bbx:related}
    {\usebibmacro{related:init}%
     \usebibmacro{related}}
    {}%
  \usebibmacro{finentry}}
```

Figura 5: Driver di biblatex per il tipo di voce bibliografica `article`.

figura 4, la prima voce è in realtà il risultato dell'assemblaggio di diverse voci bibliografiche distinte, indicanti i singoli volumi. Il file `.bib` riporta:

```
@Set{clagett_1964,
  entryset = {clagett_1964_0,clagett_1964_1,
              clagett_1964_2,clagett_1964_3,
              clagett_1964_4,clagett_1964_5},
}
@book{clagett_1964_0,
  author =  {Marshall Clagett},
  title =  {Archimedes in the Middle Ages},
  year =  {1964-1984},
  langid =  {english},
}
@book{clagett_1964_1,
  execute =  {\booltrue{nolabel}},
  title =  {The Arabo-Latin Tradition},
  volume =  {I},
  year =  {1964},
  publisher =  {The University of Wisconsin Press},
  location =  {Madison},
}
...
```

Il record bibliografico "virtuale" `clagett_1964`, di tipo `@Set`, raccoglie più record che devono comparire in bibliografia come un'unica voce, pur essendo entità logicamente separate. Il record `clagett_1964_1` (e tutti quelli che seguono, appartenenti allo stesso `Set`) ha il campo speciale `execute` che contiene codice da eseguire al momento della costruzione della voce bibliografica: in questo caso imposta a `true` il booleano `nolabel` che indica a biblatex di non far precedere l'etichetta al corpo della voce bibliografica. Utile in casi come questo in cui esiste già un'etichetta iniziale per l'opera nel suo complesso.

4.5 Altre impostazioni

Il preambolo conteneva anche altre impostazioni, alcune volte a facilitare la consistenza, da un punto di vista tipografico, dei vari contributi senza dover necessariamente ricorrere alla modifica dei testi stessi. Per esempio, il pacchetto fnpct (Niederberger, 2016) permette di "spostare" una nota a piè di pagina *dopo* la punteggiatura, anche se nel codice la precede.

Qualche parola in più va spesa a proposito del trattamento delle illustrazioni a carattere geometrico e del testo a fronte.

4.5.1 Illustrazioni geometriche

In alcuni casi è stato necessario, o quanto meno conveniente, ridisegnare alcune figure a carattere geometrico in modo da rendere più chiaro il grafico e non avere problemi di risoluzione. Queste figure sono state ridisegnate con Ti*k*Z, il quale permette di avere una stretta integrazione con il documento a livello di caratteri usati. In particolare, trattandosi per la maggior parte di disegni geometrici, il pacchetto tkz-euclide (Matthes, 2011) ha fornito un'interfaccia conveniente per trattare le costruzioni geometriche. Nella figura 6 è mostrato uno dei grafici ridisegnati con Ti*k*Z per il volume degli atti, corredato del codice per generarlo. La classe standalone (Scharrer, 2018) è stata usata per poter compilare la figura sia autonomamente che come parte del documento senza dover modificare il codice. Questo consente di poter controllare rapidamente il risultato durante la fase di disegno o di correzione, senza dover compilare l'intero documento.

4.5.2 Testo a fronte

Per finire, alcuni contributi presentavano un raffronto tra versioni differenti di uno stesso testo, o tra un testo e una sua traduzione. Abbiamo deciso, dove era possibile e opportuno, di affiancare i testi raffrontati, sempre su due colonne della stessa pagina, trattandosi di brani piuttosto corti. A questo scopo è stato usato il pacchetto paracol (Nakashima, 2018), a mio giudizio il più versatile tra quelli che consentono di porre due o più testi a confronto. Il pacchetto è molto semplice da usare, anche se permette di fare cose più complicate. In una situazione tipica, è sufficiente racchiudere il testo all'interno dell'ambiente paracol e usare `\switchcolumn` ogni volta che si cambia colonna o `\switchcolumn*` nel caso in cui si vogliano sincronizzare le due colonne, cioè allineare i capoversi. Nella figura 7 sono mostrati alcuni brani in greco antico, affiancati dalla loro traduzione.

5 Considerazioni finali

In questo articolo ho voluto condividere la mia esperienza nella composizione di un volume di atti di convegno che ha presentato, durante la lavorazione, diverse sfide da cui ritengo che LaTeX sia uscito molto bene e abbia dimostrato di poter essere usato nella produzione non solo di materiale in cui sia prevalente o fortemente presente del testo a carattere matematico. Anche in ambito umanistico (il convegno preso in considerazione, benché trattasse di storia della scienza, presentava più affinità con la filologia che con le "scienze esatte") può rivelarsi un'ottima scelta e, in combinazione con altri strumenti, costituire un ambiente di lavoro confortevole. In particolare l'ambiente di lavoro assemblato per questo progetto è stato un ambiente costruito per intero su software libero. L'ambiente si è rivelato affidabile e flessibile, consentendo di superare senza troppe difficoltà, per esempio, i vari ripensamenti nel formato delle voci bibliografiche. Quello che si vede nella versione finale del volume è solo l'esito finale di una serie di adattamenti, man mano che le circostanze facevano venire alla luce esigenze nuove. Penso soprattutto alla gestione del rapporto tra opere collettanee e singoli contributi, alla fine risolto con l'uso del campo `xref`; e al formato dell'etichetta che ha subito diverse modifiche prima della forma finale.

LaTeX è, insieme ai suoi pacchetti, un programma in continuo sviluppo. È normale quindi che alcune delle soluzioni adottate per questo lavoro non siano immediatamente replicabili per lavori attuali. L'esperienza acquisita rimane comunque valida nell'indicare in quali direzioni andranno cercate in futuro soluzioni più al passo con i tempi.

Ho tralasciato, per ragioni di spazio e di minore interesse, alcuni aspetti più marginali di questo lavoro, come la gestione delle figure con testo affiancato o la composizione di testo in versi. Ho tralasciato anche la questione di come produrre un indice dei nomi perché per questo progetto non è stato necessario adottare soluzioni meritievoli di un'esposizione dettagliata.

Alla fine dell'articolo ho voluto aggiungere una sorta di galleria di esempi che mostri alcuni degli aspetti che non sono stati trattati o che sono stati trattati solo sommariamente nel testo, o che diano in ogni caso un'idea del risultato finale (figure 8– 14). Spero che siano sufficienti a suscitare interesse in chi dovesse trovarsi ad affrontare le stesse problematiche che ho dovuto affrontare io.

Riferimenti bibliografici

Caye, P., Nanni, R. e Napolitani, P. D. (a cura di) (2015). *Scienze e rappresentazioni. Saggi in onore di Pierre Souffrin*, volume 5 di *Biblioteca Leonardiana — Studi e Documenti*. L.S. Olschki, Firenze.

Charette, F. e Reutenauer, A. (2018). «Polyglossia: an alternative to Babel for XeLaTeX and LuaLaTeX». Leggibile con `texdoc polyglossia`.

```
\documentclass{standalone}
\usepackage{fontspec}
\usepackage{tkz-euclide}
\usetkzobj{all}
\begin{document}
\begin{tikzpicture}[font=\footnotesize\sffamily,
  label style/.style={font=\footnotesize\sffamily},
  scale=1.2]
  \tkzDefPoint(0,0){A}
  \tkzDefPoint(7,0){B}
  \tkzDefPoint(10,0){c}
  \tkzDefPoint(10,3.5){C}
  \tkzDrawSegment(A,c)
  \tkzDrawSegment(A,C)
  \tkzDefLine[orthogonal=through B,
              /tikz/overlay](A,B)
  \tkzGetPoint{b}
  \tkzInterLL(A,C)(B,b)
  \tkzGetPoint{L}
  \tkzDrawSegment(B,L)
  \tkzDefPointWith[orthogonal normed, K=2](L,C)
  \tkzGetPoint{T}
  \tkzDefMidPoint(L,T)
  \tkzGetPoint{O}
  \tkzDrawCircle(O,L)
  \tkzDefLine[orthogonal=through O,
              /tikz/overlay](L,T)
  \tkzGetPoint{k}
  \tkzInterLC(O,k)(O,L)
  \tkzGetPoints{G}{K}
  \tkzInterLC(B,b)(O,L)
  \tkzGetPoints{N}{n}
  \tkzDrawSegment(L,N)
  \tkzDefPointBy[reflection = over L--N](O)
  \tkzGetPoint{O'}
  \tkzDefLine[orthogonal=through O,
              /tikz/overlay](N,L)
  \tkzGetPoint{o}
  \tkzDefCircle(O',L)
  \tkzInterLC[/tikz/overlay](O,o)(O',L)
  \tkzGetPoints{R}{R'}
  \tkzLabelPoint[below left](A){A}
  \tkzLabelPoint[below right](B){B}
  \tkzLabelPoint[above right](C){C}
  \tkzLabelPoint[below right](L){L}
  \tkzLabelPoint[below left](K){K}
  \tkzLabelPoint[above right](G){G}
  \tkzLabelPoint[above right](N){N}
  \tkzLabelPoint[left](R){R}
  \tkzClipCircle(O,L)
  \tkzDrawCircle(O',L)
\end{tikzpicture}
\end{document}
```

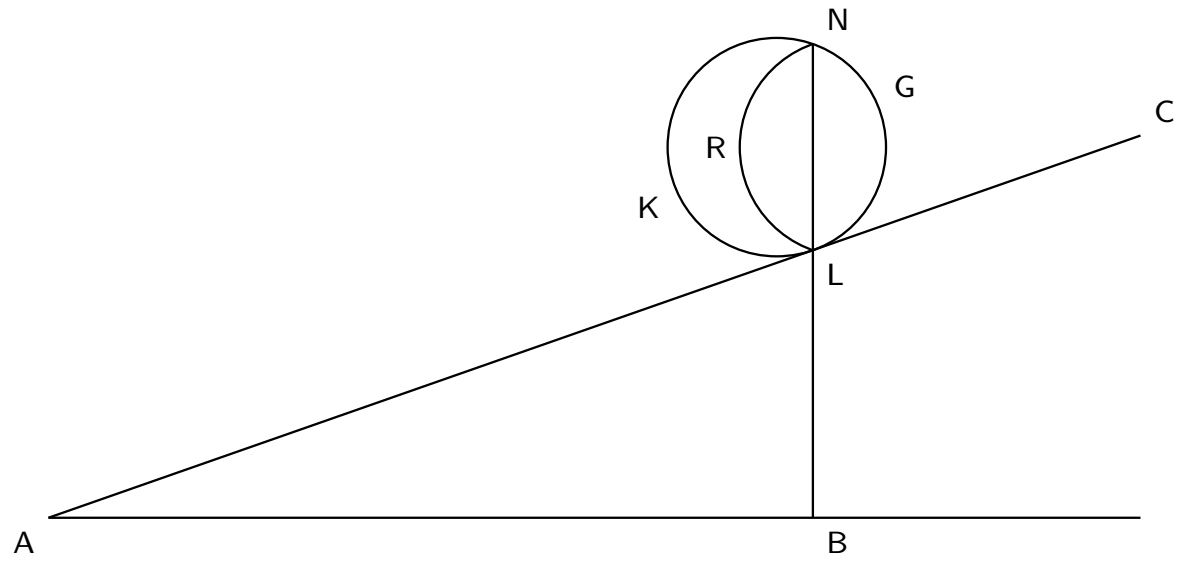


Figura 6: Disegno geometrico ottenuto con il pacchetto tkz-euclide. In alto il codice, in basso il risultato.

Cael. IV 4, 311b9-10: Σημεῖον δ' ὅτι ἕλκει πλεῖον ὁ πε- φυσημένος ἀσκὸς τοῦ κενοῦ	C'è segno, perché preme più l'otre pieno di quello vuoto.
Ph. IV 8, 216a27-29: ὥσπερ γὰρ ἐὰν ἐν ὕδατι τιθῇ τις κύβον, ἐκστήσεται τοσοῦτον ὕδωρ ὅσος ὁ κύβος, οὕτω καὶ ἐν ἀέρι· ἀλλὰ τῇ αἰσθήσει ἄδηλον.	Come infatti qualora si ponga in acqua un certo cubo, esce tanta acqua quanto è il cubo, e così anche in aria, anche se al senso ciò non appare chiaro.
Ph. IV 8, 216b18-20: ὁ γὰρ ἀὴρ ἔστιν τι, οὐ δοκεῖ δέ γε – οὐδὲ τὸ ὕδωρ, εἰ ἦσαν οἱ ἰχθύες σιδηροῖ·	L'aria è infatti un certo qualcosa, e pure non pare, e neppure ‹parrebbe› l'acqua, se i pesci fossero di ferro
Ph. IV 9, 217a2-3: ἀλλ' ὥσπερ οἱ ἀσκοὶ τῷ φέρεσθαι αὐτοὶ ἄνω φέρουσι τὸ συνεχές, οὕτω τὸ κενὸν ἄνω φέρει	Ma come gli otri spostando sé stessi in alto, spostano ciò che è loro continuo: in questo modo il vuoto muoverebbe verso l'alto
Ph. VIII 5, 255b25-26: οἷον ὁ τὸν κίονα ὑποσπάσας ἢ ὁ τὸν λίθον ἀφελὼν ἀπὸ τοῦ ἀσκοῦ ἐν τῷ ὕδατι	Come chi leva di sotto la colonna o chi toglie il sasso dall'otre nell'acqua

Figura 7: Esempio di testo a fronte. A sinistra i brani in greco, a destra la traduzione.

de Bari, O. (2006). «GNU Emacs e AUCTEX per LATEX». *ArsTEXnica*, (2), pp. 60–64. URL http://www.guitex.org/home/numero-2.

Gilbert Moschou, A. (2016). «The mathspec package. Font selection for mathematics with XELATEX». Leggibile con texdoc mathspec.

Giordano, M., Iovino, O. e Leccardi, M. (2013). «Guida pratica all'uso di GNU Emacs e AUCTEX». v.1.0. Guida GUIt. URL http://www.guitex.org/home/images/doc/GuideGuIT/guidaemacsauctex.pdf.

Git (2018). «Git». URL https://git-scm.com/.

Gregorio, E. (2010). «Installare TEX live 2010 su ubuntu». *ArsTEXnica*, (10), pp. 7–13. URL http://www.guitex.org/home/numero-10.

Just, H. (2018). «Writer2LaTeX». URL http://writer2latex.sourceforge.net/.

Lehmann, P. (2018). «The biblatex Package». Leggibile con texdoc biblatex.

Matthes, A. (2011). «tkz-euclide». Leggibile con texdoc tkz-euclide.

Nakashima, H. (2018). «Package paracol. Yet Another Multi-Column Package to Typeset Columns in *Parallel*». Leggibile con texdoc paracol.

Napolitani, P. D. e Gautero, J.-L. (2006). «En mémoire de pierre souffrin». *Revue d'histoire des sciences*, **59** (2), pp. 187–196. URL https://doi.org/10.3917/rhs.592.0187.

Niederberger, C. (2016). «FNPCT». URL https://bitbucket.org/cgnieder/fnpct/. Leggibile con texdoc fnpct.

Robertson, W. (2018). «The fontspec package. Font selection for XETEX and LuaTEX». URL http://wspr.io/fontspec/. Leggibile con texdoc fontspec.

Scharrer, M. (2018). «The standalone Package». Leggibile con texdoc standalone.

Souffrin (2012). «Science et Représentations. Colloque International en mémoire de Pierre Souffrin». URL http://www.bibliotecaleonardiana.it/convegnosouffrin/home.html.

Wilson, P. e Madsen, L. (2018). «The memoir class». Leggibile con texdoc memoir.

▷ Massimiliano Dominici
mlgdominici at gmail dot com


BIBLIOTECA
LEONARDIANA
STUDI E DOCUMENTI
5

SCIENZE E
RAPPRESENTAZIONI
SAGGI IN ONORE DI PIERRE SOUFFRIN

Atti del convegno internazionale
Vinci, Biblioteca Leonardiana, 26-29 settembre 2012

a cura di
Pierre Caye, Romano Nanni e Pier Daniele Napolitani

L.S.O

LEO S. OLSCHKI EDITORE
MMXV

Figura 8: Frontespizio del volume.

STEFANO GATTEI

THE PHOENIX AND THE ARCHITECT: THE FRONTISPIECE OF KEPLER'S «TABULAE RUDOLPHINAE»*

> Tycho possesses the best observations, which constitute, as it were, the material for the erection of this structure; he also has workers, and everything else which one might desire. He only lacks an architect who uses all these according to his own plan.
>
> JOHANNES KEPLER[1]

In the «Preface» to his major work on optics, *Ad Vitellionem paralipomena* (1604), Johannes Kepler explicitly likened astronomy to a house, and the astronomer's task to that of an architect:

You see, dear reader, the position occupied by Tychonic astronomy, that is, the truest and most accurate astronomy. He collected most ample material for a future building in the observation books; he showed the soundness of that material in the *Mechanica*; he laid two very solid foundations for the house […]: through the Catalogue of Fixed [Stars], described most accurately and truly, which played the role of the best cement and will serve to glue together the material of the observations; and through the theory

---

* A preliminary version of this article was published in the proceedings of an international conference on scientific images held in Pisa in October 2008: see GATTEI 2009. I am developing this study into a book, under contract with Oxford University Press.

[1] JKGW XIX, n. 2.1, p. 37; see also Tycho Brahe's letter to Ján Jesenský, dated 29 March/8 April 1600, in JKGW XIV, n. 161, p. 114. In the *Apologia pro Tychone contra Ursum*, which he left unfinished at the time of Tycho's death in 1601, Kepler wrote: «In architecture it is not usual for everything – limestone, quarry-stone, nails, bars and windows – to be asked from a single workshop; nor is one and the same person the architect, the quarryman, the carpenter, the blacksmith, and the cabinet-maker. The same holds for the business of astronomy. For he who devotes the full strength of his mind to establishing, by inference from phenomena that occur at different times in the sky, some hypotheses that represent the form of the universe, I regard as the architect. His task, however, is so great that it is impossible for him to seek out everything by himself. So he gets observations themselves, either all or some of them, from others […]»: KEPLER 2008, p. 317; see also KEPLER 1984a, pp. 185-186. On the difference of role and importance between the architect and his co-workers, see Aristotle, *Metaphysics*, A, 1, 981a30-982a3 (a passage that was clearly present to Kepler when he wrote the *Apologia* and the *Paralipomena*, and which he was sure his readers needed not to be reminded of).

Figura 9: Pagina iniziale di un contributo.

beam in terms of thickness and matter mean both equal and similar in Arabic. There is a clear preference in Arabic mathematical texts for using the first for equal and the second for similar. Thus, Knorr translated them in this manner (Knorr 1982, p. 139). In the given context it is clear though that similarity is not meant literally, but in the sense of having the same property. This ambiguity reflects the use of ἴσος and ὁμοίος for respective concepts in Greek.

5.2. *Investigation 2*

Liber de Canonio, Proposition II

Si fuerit proportio ponderis in termino minoris portionis suspensi, ad superhabundantiam ponderis maioris portionis ad minorem, sicut proportio longitudinis totius canonii ad duplam longitudinis minoris portionis, erit canonium parallelum epipedo orizontis (Moody & Clagett 1952, p. 66).

If the proportion of the weight suspended at the end of the smaller portion to the surplus of the weight of the greater portion to the smaller will be like the proportion of the length of the entire beam to the double of the length of the smaller portion, the beam will be parallel to the surface of the horizon (Cf. Moody & Clagett 1952, p. 67).

MS Beirut, *ziyāda*, Proposition 4

اذا كان عمود متساوي الغلظ متشابه الجوهر وقسم بقسمين مختلفين وعلق بنقطة طرف القسم الاقصر ثقل وجعلت نسبة الثقل الى ثقل فضل القسم الاطول على ثقل القسم الاقصر كنسبة نصف طول العمود كله الى طول القسم الاقصر فان العمود يعتدل على موازاة الافق.

(Knorr 1982, p. 154).

If there is a beam, (which is) equal in itself in thickness, equal in itself in substance and partitioned in two different parts and (if) a weight is suspended at the end of the shorter part and the ratio of the weight to the weight of the surplus of the longer part over the weight of the shorter part is made like the ratio of half of the length of all of the beam to the length of the shorter part, then the beam equilibrates itself in parallelness to the horizon.

Again, the content of both theorems is the same and the two enunciations are similar, but not identical. Their difference is greater than in the previous case, because the *Liber de canonio* does not repeat the description of the properties of the beam and the suspended weight and thus has to integrate the latter into the description of the proportion. It differs from the *ziyāda* also in regard to the placement of the term *weight* in the description of the second term of the proportion. The *Liber de canonio* uses the term only once between *superhabundatiam* and *maioris*. The *ziyāda* uses it twice, once before the surplus and once before the shorter part. While the formulation of the *Liber de canonio* is imprecise, but comprehensible, the formulation of the *ziyāda* is comprehensible, but false. It is most likely the result of a scribal error as

Figura 10: Testo arabo con andamento da destra a sinistra.

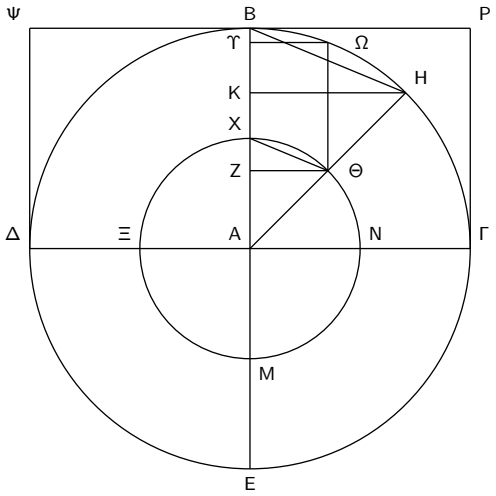


Figura 2: Da KRAFFT 1970, p. 27.

re di Archimede è molto interessante: infatti, per il modo con cui è costruito, gli *Equiponderanti* sono una sorta di trattato di geometria meccanica costruito per realizzare la quadratura della parabola. Questo è un indice, di per sé significativo, della rigida delimitazione disciplinare; per poter essere utilizzata come strumento per la soluzione di problemi geometrici, la meccanica doveva essere trasformata da disciplina fisico-matematica in disciplina matematica.

6. I PRINCIPI DELLA MECCANICA NELLE «QUESTIONI MECCANICHE» PSEUDOARISTOTELICHE E IN VITRUVIO

Anche i principi delle *Questioni meccaniche* sono formulati nella forma di cerchi concentrici diseguali, ma con modalità e funzioni completamente diverse dai trattati di meccanica tradizionali (vedi Fig. 2). Non ci sono presupposti di sospensione di pesi, ma il principio emerge naturalmente dall'operazione di costruzione dei cerchi mediante un compasso, o più probabilmente, da una cordicella fissata ad un punto e collegata a una asta rigida che traccia i cerchi. In questo modo la forza (ἰσχύς), l'impulso iniziale che determina la rotazione, si compone con il movimento naturale (ῥοπή) e avviene che i punti del raggio maggiore si muovano più velocemente di quelli del minore in quanto la componente contro natura del movimento costringe di più verso il centro il

Figura 11: Pagina con illustrazione a carattere geometrico.

sono lunghe, e, con tutto ciò, il peso non si moverà se non quanto è la lunghezza di una sola di esse: il che sia detto per avvertimento e confermazione di quello che più volte si è di già detto, cioè che con qual proporzione si diminuisce la fatica del movente, se gli accresce all'incontro lunghezza del viaggio (GALILEI 2002, v. l. ll. 756-766).

Non è difficile rilevare che queste erano state proprio le stesse ragioni esposte da Erone per dimostrare che anche nel caso del treno di taglie (Fig. 9) era valido il *principio di compensazione*; l'unica differenza è che ogni taglia costituente detto treno è costituita da cinque pulegge, laddove la taglia di Galileo era costituita da quattro pulegge:

È chiaro che con questa macchina ci sarà un ritardo perché il processo avviene allo stesso modo. Poiché la forza in δ, che è di 200 talenti, solleva il peso da β a γ, allora deve avvolgere i cinque capi di corde attorno alle cinque pulegge per l'ammontare della distanza tra i punti β e γ, mentre la forza in η deve avvolgere le cinque corde cinque volte.

Se ora assumiamo le lunghezze $\beta\gamma$ e $\varepsilon\zeta$ uguali tra loro, allora, avvolgendo una delle corde per la lunghezza $\beta\gamma$, saranno avvolte cinque corde per la lunghezza $\varepsilon\zeta$, perché, affinché il peso percorra la distanza tra β e γ, cinque corde devono essere avvolte per l'ammontare della distanza $\beta\gamma$. Allora il rapporto dei tempi è uguale al rapporto [inverso] delle forze motrici (ERONE 1988, II, 24, pp. 154-155).

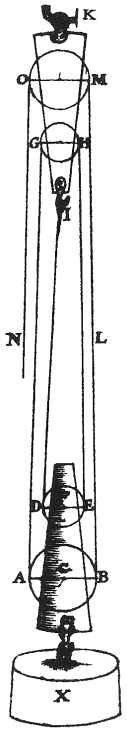


Figura 11: G. Galilei, *Le Mecaniche*. La taglia.

Potrei far vedere che analoghe considerazioni accompagnano le conclusioni della trattazione della leva e dell'asse nella ruota dei due autori. Mi preme invece fermarmi su un ulteriore aspetto della trattazione galileiana che mi sembra trovi riscontro in quella di Erone.

Figura 12: Pagina con illustrazione immersa nel testo.

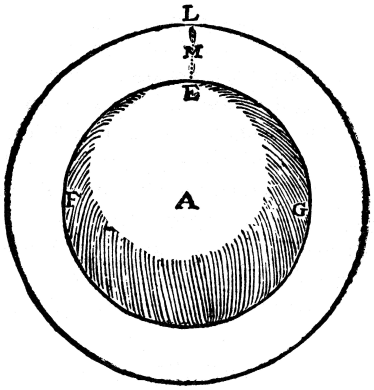


Figure 6: Dessin de Gilbert illustrant la chute verticale sur une Terre en rotation.

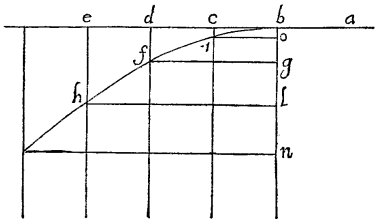


Figure 7: La chute parabolique dans les *Discorsi* de 1638 (GALILEI 1968, VIII, p. 272).

t^2, il fera appel, dans les *Discorsi*, à une composition des mouvements pour déterminer la loi parabolique du projectile.

SALVIATI: Ora possiamo ripigliare il testo, per vedere in qual maniera ei vien dimostrando la sua prima proposizione, dove egli intende di provarci la linea descritta dal mobile grave, che mentre ci descende con moto composto dell'equabile orizontale e del naturale descendente, sia une semiparabola (GALILEI 1968, VIII, p. 272).

SALVIATI: Nous pouvons donc reprendre le texte pour voir comment il [l'Académicien] démontre sa première proposition, dans laquelle il entend prouver que la trajectoire décrite par un mobile pesant, alors qu'il descend d'un mouvement composé d'un mouvement horizontal uniforme et du mouvement naturel de chute, est une demi-parabole (GALILEI 1970, pp. 208-209).

Grâce à cette démonstration, Galilée déduit mathématiquement la forme de la trajectoire au moyen de la composition d'un mouvement inertiel uniforme et d'un mouvement de chute vertical dont il a découvert la loi expérimentalement. Plus que Gilbert ou que Kepler c'est cette démonstration là qui va stimuler la réflexion de Newton.

3.3. *Philippe van Lansberg, une critique étonnante*

En 1619, Philippe van Lansberg (1561-1632) élabore une critique du système de Tycho Brahe dans ses *Progymnasmatum astronomiae restitutae*. Au chapitre XXIII, intitulé *Quod motus annuus qui apparet in Sole revera sit motus Terrae*, il explique le principe de relativité que l'on qualifie d'optique à la

Figura 13: Altra pagina con illustrazione immersa nel testo.

Paola Manni

SULLA TERMINOLOGIA DELLE MACCHINE IN LEONARDO: TRADIZIONE, INNOVAZIONE E SVILUPPI FUTURI*

Qui si dimostra la natura della vite e di sua lieva,
e chome ella debbe più tosto ess(er)e adop(er)ata <in is>
in tirare che in ispingiere. E chom'ella fa più for-
ça a essere senplice che doppia, e sottile che grossa,
essendo mossa da pari lungeça di lieva e pari força.
E chosì si farà un pocho di discorso in qua(n)ti modi si
pò adop(er)are, e di qua(n)te sorte si pò fare viti sança
fine. E qua(n)ti moti son fatti sança vite, che fa(n)-
no p(r)opio ofitio di vite. E in che modo la vite
sança fine s'achonpagni cholle rote dentate, e
chome molte viti si debono insieme adop(er)are.
E ssi dirà della natura delle sue madri, e sse so(n)
più utili cho· molti denti o nno. E si dirà delle
viti retrose e delle viti che p(er) un medesimo ti-
rare spingano e ttirano il peso, e di viti che
p(er) una sola volta che se le dia, farà fugire la sua
madre molte delle sue volte circulari. E così
moltissimi sua effetti, e varie fatiche, e fforteçe,
e tardità, e p(r)esteçe. E ssi prov(er)rà ragio(n)e[1] <di ut>
di tutti loro ofiti e nature, e materie, e llieve,
e utilità. E ssi dirà in che modo si debbono fare,
e del modo del metterle in op(er)a;
e di chi è stato inganato p(er) no(n) cognosscer lor natura.
E ttali strume(n)ti si figurera(n)no in gra(n) parte sança
le loro armadure, o altra cosa che avessi a inpe-

* Le trascrizioni dai codici leonardiani sono fatte seguendo le norme stabilite da Arrigo Castellani per l'edizione dei testi medievali, già utilizzate in Manni 2008 e in Manni & Biffi 2011. Alle pagine introduttive di quest'ultimo (pp. xxxi-xxxii) si rimanda per una loro esposizione dettagliata e ulteriori riferimenti bibliografici. Nel caso di citazioni brevi inserite nel corpo del testo, si eliminano le parentesi tonde che segnalano lo scioglimento delle abbreviazioni. Con la sigla Madrid I si indica il primo codice di Madrid (Biblioteca Nacional de España, cod. 8937).

[1] La *e* non chiara, corretta su altra lettera.

Figura 14: Testo in versi numerati.

Axessibility: creating PDF documents with accessible formulae

D. Ahmetovic, T. Armano, M. Berra, C. Bernareggi, A. Capietto, S. Coriasco, N. Murru, A. Ruighi

Sommario

I documenti PDF contenenti formule generati da LaTeX non sono solitamente accessibili mediante tecnologie assistive per persone con disabilità visive (i.e., screen reader e barre Braille). Il pacchetto LaTeX `axessibility.sty` da noi sviluppato risolve questo problema, permettendo di creare documenti PDF in cui le formule vengono lette da tali tecnologie assistive. Infatti, vengono generati automaticamente dei commenti nascosti nel documento PDF (mediante l'attributo `/ActualText`) in corrispondenza di ogni formula. Tale testo alternativo risulta nascosto nel documento PDF, ma gli screen reader Jaws, NVDA e VoiceOver vi accedono correttamente. Inoltre, abbiamo creato dei dizionari (in inglese e italiano) per NVDA e Jaws che forniscono la lettura delle formule in linguaggio naturale nel caso in cui l'utente non conosca i comandi LaTeX. Il pacchetto non genera PDF/UA.

Abstract

PDF documents containing formulae generated by LaTeX are usually not accessible by assistive technologies for visually impaired people (i.e., by screen readers and braille displays). The LaTeX package `axessibility.sty` that we developed manages this issue, allowing to create PDF documents where the formulae are read by these assistive technologies, since it automatically generates hidden comments in the PDF document (using the `/ActualText` attribute) in correspondence to each formula. This actual text is hidden in the PDF document, but the screen readers Jaws, NVDA and VoiceOver read it correctly. Moreover, we have created NVDA and Jaws dictionaries (in English and in Italian) that provide the reading in the natural language in the case that the user does not know the LaTeX commands. The package does not generate PDF/UA.

1 Introduction

In this paper, we describe `axessibility.sty`, a LaTeX package which allows to automatically generate a PDF document with formulae accessible by assistive technologies for visually impaired people. Assistive technologies (screen readers and braille displays) perform satisfactorily with regard to digital documents containing text, but they still have a long way to go as far as formulae and graphs are concerned. A comprehensive overview about this problem can be found in Archambault *et al.* (2007) and Armano *et al.* (2014).

Many studies have been conducted in order to improve the accessibility of digital documents with mathematical contents. For instance, MathPlayer ensures accessibility of formulae inserted by using MathType in Word documents Soiffer (2005). Another way for creating accessible mathematical documents is given by the MathML language (see Bernareggi e Archambault (2007) for further information). However, accessibility of such documents is heavily affected by the versions of browsers, operating systems and screen readers, making this solution very unstable. A system used by blind people for reading and writing mathematics is the LAMBDA system (Linear Access to Mathematics for braille Device and Audio-synthesis). Mathematical language in LAMBDA is designed so that every symbol can be directly translated into words. For further details on LAMBDA we refer to Bernareggi (2010). Unfortunately, this system does not help to spread accessible digital documents, since it is only used by visually impaired people and it is not a standard for the realisation of documents by sighted people. Regarding LaTeX, assistive technologies can directly manage LaTeX documents. In this case, visually impaired people need to learn LaTeX in order to understand the commands. However, there are software which facilitate LaTeX comprehension and usability; one of them is BlindMath Pepino *et al.* (2006). Moreover, some converters from LaTeX to braille exist, see, e.g., Papasalouros e Tsolomitis (2017) and Batusic *et al.* (1998).

In general, the most widespread digital documents are in PDF format. However, in the case of mathematical contents, they are not accessible at all, since formualae are usually unreadable by screen readers because they are bidimensional as images. None of the above systems allows to directly produce accessible formulae in PDF documents. This could be possible only performing specific tasks. For instance, using the Word editor, if each formula is manually tagged by the author (by using the alternative text), such a comment will be kept when the corresponding PDF file will be

generated and it will be read by the screen reader. However, this procedure does not help to improve the presence of accessible PDF documents, since it is a very boring and time consuming method. It is very hard to think that an author performs these actions for the realisation, e.g., of a book. Currently, a standard and fast method for inserting accessible formulae into a PDF documents is still lacking despite it is a very important issue for spreading accessible digital scientific documents. In Uebelbacher *et al.* (2014) standard guidelines for accessibility of PDF documents are presented. Moreover, in Moore (2009), Moore (2014) and Borsero *et al.* (2016), an overview about accessibility of PDF documents is provided with a focus on mathematical contents.

In this paper, we show the features of the package `axessibility.sty` (whose a first version is also described in Armano *et al.* (2018)) that provides the first method for an automatised production of accessible PDF documents with mathematical contents. We would like to highlight that this package does not produces fully tagged PDF, such as the standard PDF/UA, but it allows to obtain a PDF where formulae are described using the `/ActualText` attribute.

2 Problem Statement

When a PDF document is generated starting from LaTeX, formulae are not accessible by screen readers and braille displays. They can be made accessible by inserting a hidden comment, i.e., an actual text, similarly to the case of web pages or Word documents. This can be made, e.g., by using the LaTeX package `pdfcomment.sty` or using an editor for PDF files like Adobe Acrobat Pro. In any case, this task must be manually performed by the author and it is surely inefficient, since the author should write the formulae and, in addition, insert a description for each formula. Note also that the package `pdfcomment.sty` does not allow to insert special characters like *backslash*, *brace*, etc, in the comment. Moreover, with these solutions, the reading is bothered, since the screen reader reads incorrectly the formula and then the correct comment of the formula.

In Figure 1, we show the PDF document generated from the following LaTeX code containing a simple formula with a comment manually inserted.

```
\documentclass{article}
\documentclass[a4paper]{article}
\usepackage{pdfcomment}
\begin{document}
A simple formula:
\begin{equation}
\pdftooltip{
        \frac{1 + \sqrt{5}}{2}
        }{
        begin fraction numerator 1 +
            square root of 5 over 2
            end fraction
        }
\end{equation}
\end{document}
```

When the screen reader accesses the PDF document, the formula will be read

> square root 1 plus 5 2 begin fraction numerator 1 plus square root of 5 over 2 end fraction

i.e., before reading the correct comment

> begin fraction numerator 1 plus square root of 5 over 2 end fraction

the screen reader reads incorrectly the formula

> square root 1 plus 5 2.

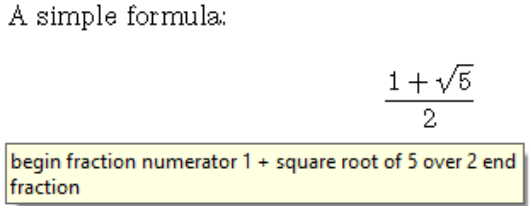


Figure 1: PDF document generated using the package `pdfcomment.sty`

There are also some LaTeX packages that try to improve the accessibility of PDF documents produced by LaTeX. In particular the packages `accsupp.sty`[1] and accessibility_meta.sty[2] has been developed in order to obtain tagged PDF documents. However, both packages do not solve the problem of the accessibility of formulae.

3 `axessibility.sty` LaTeX package

Our package, named `axessibility.sty`, solves the problem described in the previous section. It achieves that by inserting a hidden comment in the PDF file corresponding to any given formula. This comment, named `/ActualText`, contains the original LaTeX commands used to generate the formula. The hidden comment is read by screen readers and braille displays instead of the `ASCII` representation of the formula, which is often incorrect.

3.1 Usage

Authors that would like to create an accessible PDF document for visually impaired people only need to include the `axessibility.sty` package into the preamble of their LaTeX project. Mathematical environments will then automatically produce the `/ActualText` content and include it in the produced PDF file.

1. available at https://ctan.org/pkg/accsupp
2. available at https://github.com/AndyClifton/AccessibleMetaClass

We have treated the most used environments for inserting formulae, i.e., `equation`, `equation*`, `\[`, `\(`. Hence, any formula inserted using one of these environments is accessible in the corresponding PDF document. Additionally, the package enables to copy the formula LaTeX code from the PDF reader and paste it elsewhere.

Note that, to preserve the compatibility with Acrobat Reader, our package discourages the use of the underscore character `_` which is not correctly read using screen readers in combination with this PDF reader. Alternatively, we suggest to use the equivalent command `\sb`.

In-line and display mathematical modes ($, $$) are not supported in this version of the package. However external scripts provided as companion software can also address these use cases.

If we use the package `axessibility.sty` applied to the previous example, we obtain the following LaTeX code.

```
\documentclass[a4paper,11pt]{article}
\usepackage{axessibility}
\begin{document}
    A simple formula:
    \begin{equation}
        \frac{1 + \sqrt{5}}{2}
    \end{equation}
\end{document}
```

We observe that, in this case, the author has to write the formula without adding anything else. Moreover, inside the source code of the PDF file, we find an `/ActualText` tag with the LaTeXcode inside, automatically generated by the `axessibility.sty` package.

```
/S/Span<</ActualText(\040\040\\
    frac\040{1\040+\040\\sqrt
    \040{5}}{2}\040)>>BDC
```

The screen reader will read correctly the LaTeX command `\frac {1+ \sqrt {5}}{2}`. Moreover, we have created JAWS and NVDA dictionaries that provide the reading in the natural language in the case that the user does not know the LaTeX commands.

3.2 Technical Overview

`axessibility.sty` first defines a pair of internal commands (`\BeginAxessible` and `\EndAxessible`) which redefine `\BeginAccSupp` and `\EndAccSupp` as follows:

```
\newcommand*{\BeginAxessible}[1]{%
  \begingroup
    \setkeys{ACCSUPP}{#1}%
    \edef\ACCSUPP@span{%
     /S/Formula<<%
        \ifx\ACCSUPP@Alt\relax
        \else
          /Alt\ACCSUPP@Alt
        \fi
        \ifx\ACCSUPP@ActualText\relax
        \else
          /ActualText\
              ACCSUPP@ActualText
        \fi
      >>%
    }%
    \ACCSUPP@bdc
    \ACCSUPP@space
  \endgroup
}
```

Precisely, `\BeginAxessible` adds a hidden comment that starts with `/S/Formula` instead of `/Span`.

```
\newcommand*{\EndAxessible}{%
  \begingroup
    \ACCSUPP@emc
  \endgroup
}
```

The second building block of this package is the wrapper. This routine takes the LaTeX code inside the formula, removes the tokens and passes it to `\BeginAxessible`.

```
\long\def\wrap#1{
\BeginAxessible{method=escape,
    ActualText=\detokenize\expandafter
    {#1}, Alt=\detokenize\expandafter
    {#1} }
 #1
\EndAxessible%
}
```

Finally, using the wrapper, we can re-define the mathematical environments using the command above. Here is an example using `equation`.

```
\renewenvironment{equation}{%
 \incr@eqnum
  \mathdisplay@push
  \st@rredfalse \global\@eqnswtrue
 \mathdisplay{equation}%
  \collect@body\wrap\auxiliaryspace}{%
  \endmathdisplay{equation}%
  \mathdisplay@pop
  \ignorespacesafterend
}
```

4 Conclusions and Future Work

We have developed a LaTeX package that automatically generates comments to formulae when the PDF document is produced by LaTeX. The comments are hidden in the PDF document and they contain the LaTeX commands that generate the formulae. In this way, an accessible PDF document containing formulae is generated. Indeed, screen readers are able to access the comment when processing a formula and reading it. Moreover, we have created JAWS and NVDA dictionaries that provides the reading in the natural language in the case that the user does not know the LaTeX commands.

There are a few issues that are yet to be solved with a pure LaTeX solution. Namely,

- In-line and displayed mathematical environments delimited with `$, $$`,
- User-defined Macros,
- Multi-line environments such as `\align` and `\eqnarray`.
- Semantic description of formulae.
- PDF/UA.

We have resolved the first two problems using an external script – `axesscleaner.py` – coded in `Perl` and `Python`. The script also beautifies the LaTeX file and removes all the *underscore* characters `_`, replacing those with `\sb`. Using this solution we are now able to apply `axessibility.sty` to entire textbooks that were written without using the package in the first place.

Multi-line environments are going to be treated using a LaTeX solution that is currently in the test phase. Concerning the last two problems a more in-depth research is in order. The authors are currently initiating the investigation to address these issues as a future work.

5 Acknowledgements

The authors wish to thank the bank foundation 'Fondazione Cassa di Risparmio di Torino', Leo-Club (Biella, Italy) and the several volunteers with visual impairment who provided their fundamental contribution.

References

Archambault, D., Stoger, B., Fitzpatrick, D. e Miesenberger, K. (2007). «Access to scientific content by visually impaired people». *Upgrade*, **2**, p. 14.

Armano, T., Capietto, A., Illengo, M., Murru, N. e Rossini, R. (2014). «An overview on ict for the accessibility of scientific texts by visually impaired students». *Proceedings Conference SIREM-SIE-L*, **2014**, pp. 119–122.

Armano, T., Capietto, A., Coriasco, S., Murru, N., Ruighi, A. e Taranto, E. (2018). «An automatized method based on latex for the realization of accessible pdf documents containing formulae». *Computers Helping People with Special Needs, Lecture Notes in Computer Science*, **10896**, pp. 583–589.

Batusic, M., Miesenberger, S., K. e LaBraDoor, B. (1998). «a contribution to making mathematics accessible for the blind». In *Proceedings of 6th International Conference on Computers Helping People with Special Needs*. ICCHP 1998, Oldenbourg, Wien, Munchen.

Bernareggi, C. (2010). «Non-sequential mathematical notations in the lambda system». *Computers Helping People with Special Needs, Lecture Notes in Computer Science*, **6180**, pp. 389–395.

Bernareggi, C. e Archambault, D. (2007). «Mathematics on the web: emerging opportunities for visually impaired people». In *07 Proceedings of the 2007 international cross-disciplinary conference on Web accessibility (W4A), Banff (Canada),*, a cura di P. W. 4A. pp. 108–111.

Borsero, M., Murru, N. e Ruighi, A. (2016). «Il latex come soluzione al problema dell'accesso a testi con formule da parte di disabili visivi». *ArsTeXnica*, **22**, pp. 12–18.

Moore, R. (2009). «Ongoing efforts to generate tagged pdf using pdftex». *TUGboat*, **30**, pp. 170–175.

— (2014). «Pdf/a-3u as an archival format for accessible mathematics». In *Watt*, a cura di M. Stephen, J. H. Davenport, A. P. Sexton, P. Sojka e J. Urban, CICM, Lecture Notes in Computer Science 8543, pp. 184–199.

Papasalouros, A. e Tsolomitis, A. A. (2017). «direct tex-to-braille transcribing method». *Journal of Science Education for Students with Disabilities*, **20** (5).

Pepino, A., Freda, C., Ferraro, F., Pagliara, S. e Zanfardino, F. (2006). «Blindmath». In *a New Scientific Editor for Blind Students*, a cura di C. Helping, People with Special Needs, Lecture Notes in Computer Science 4061, pp. 1171–1174.

Soiffer, N. M. w.-b. m. a. (2005). «Assets '05: Proceedings of the 7th international acm sigaccess conference on computers and accessibility». *New York (USA)*, pp. 204–205.

Uebelbacher, A., Bianchetti, R. e Riesch, M. P. (2014). «Accessibility checker (pac 2): The first tool to test pdf documents for pdf/ua compliance». *Computers Helping People with Special Needs, Lecture Notes in Computer Science*, **8547**, pp. 197–201.

▷ D. Ahmetovic
Dipartimento di Matematica "G. Peano", Università degli Studi di Torino
dragan.ahmetovic@unito.it

▷ T. Armano
Dipartimento di Matematica "G. Peano", Università degli Studi di Torino
tiziana.armano@unito.it

▷ M. Berra
Dipartimento di Matematica "G. Peano", Università degli Studi di Torino
michele.berra@unito.it

▷ C. Bernareggi
Dipartimento di Informatica, Università di Milano
cristian.bernareggi@unimi.it

▷ A. Capietto
Dipartimento di Matematica "G. Peano", Università degli Studi di Torino
anna.capietto@unito.it

▷ S. Coriasco
Dipartimento di Matematica "G. Peano", Università degli Studi di Torino
sandro.coriasco@unito.it

▷ N. Murru
Dipartimento di Matematica "G. Peano", Università degli Studi di Torino
nadir.murru@unito.it

▷ A. Ruighi
Dipartimento di Matematica "G. Peano", Università degli Studi di Torino
alice.ruighi@unito.it

Experimenting with makeindex and Unicode, and deriving kameindex

Antoine Bossard, Keiichi Kaneko

Abstract

When writing and editing a long document such as a book or thesis, it is almost inevitable to include an index. The purpose of this part of a document is to collect several keywords that appear in the main text, listing them together with the corresponding pages numbers, in order to facilitate information search inside the document. Document typesetting is often realised with the LaTeX system, especially for professional and academic editing. The LaTeX typesetting system embeds via the *makeindex* subsystem an indexing feature, which semi-automatically generates a document's index. However, makeindex has not been developed to support Unicode and thus fails in most cases at generating indices that include multilingual entries, and especially entries of different writing systems. In this paper, we first review the, even scarce, multilingual capabilities of makeindex, before deriving *kameindex*, a solution to these internationalization issues. Importantly, kameindex replaces one component of the makeindex subsystem, thus retaining overall compatibility with makeindex. After illustrating the results achieved by kameindex, comparison with related works is conducted.

Sommario

Quando scriviamo o editiamo un documento lungo come un libro o una tesi, è quasi inevitabile includere un indice. Lo scopo di questa parte di documento è elencare una serie di parole chiave che appaiono nel testo corredandole con i corrispondenti numeri di pagina, così da facilitare la ricerca dell'informazione nel documento. La composizione di documenti avviene spesso in LaTeX, specialmente per pubblicazioni professionali e accademiche. Il sistema di composizione LaTeX incorpora uno strumento di indicizzazione grazie a *makeindex*, che genera semiautomaticamente l'inidce del documento. Purtroppo makeindex non è stato sviluppato per supportare Unicode e quindi nella maggior parte dei casi sbaglia a generare gli indici che includono voci multilingue, e specialmente voci in diversi alfabeti. In questo articolo, inizialmente recensiamo le pur scarse capacità multilingua di makeindex, prima di derivare *kameindex*, una soluzione a questi problemi di internazionalizzazione. Cosa importante, kameindex sostituisce un componente di makeindex, mantenendo quindi la compatibilità generale con makeindex. Dopo aver illustrato i risultati permessi da kameindex, faremo una comparazione con altri lavori correlati.

1 Introduction

Indexing is about gathering the keywords of a document (e.g., thesis, book) and listing them together with the corresponding page numbers in a special section – the index – usually included at the end of the document. Indices are especially useful for long documents: without such information, it would be very tedious to search inside the document. Indeed, paper documents do not provide an automated search feature as with electronic ones. The usage of the document typesetting system LaTeX is widely spread, especially throughout the academic community. Thanks to the *makeindex* subsystem, LaTeX is capable of semi-automatically generating indices. This process is semi-automatic since the writer has to manually declare index entries throughout the document.

An important and well-known shortcoming of makeindex is its lack of Unicode support. Effectively, makeindex has been developed to process ASCII files only, that is, makeindex works on a byte per byte basis. Consequently, makeindex is in most cases failing at realising multilingual indices – especially when mixing writing systems. In this paper, the first objective is to review the, even lacking, multilingual capabilities of makeindex. The second objective is to derive from the exposed makeindex issues a Unicode-capable indexing solution, *kameindex*, that will replace one component of the makeindex subsystem, thus importantly retaining compatibility with the original makeindex system.

This research is part of a wider project that aims at improving the support of the Japanese writing system by computers and information and communication technology (ICT) in general. In a previous work, we have described an unrestricted character encoding for Japanese (Bossard and Kaneko, 2018). Further advancing this subject, we consider in this paper an application – indexing in LaTeX – where Japanese support is lacking, and propose concrete solutions to address this issue. The long term objective is to support the previously proposed character encoding with this indexing application.

A large part of this paper is addressing multilingual documents. We rely on Unicode and its UTF-8

implementation (Unicode Technical Committee, 2011) for character representation (encoding) purposes. LaTeX source code mentioned hereinafter is compiled with XeLaTeX (Robertson and Hosny, 2017). Therefore, font loading is easily realised with the `fontspec` package and, for instance, its `\newfontfamily` command.

2 Reviewing makeindex capabilities

The fundamental features of makeindex are first briefly presented. Then, workarounds for multilingual (Unicode) documents are presented.

2.1 Basic features

The makeindex utility and corresponding LaTeX package (Lamport, 1987) are used to automatically generate indices from a set of index entries declared inside the source document (call the `\makeindex` command to activate indexing). An index entry is declared with the command `\index`. makeindex is then in charge of compiling the index by separating entries (per the first letter), sorting them, merging identical ones, collecting and merging page numbers, calculating page ranges and so on. Finally, the output of the resulting index is requested to the compiler with the `\printindex` command.

The `\index` command can also be used to declare sub-entries: `\index{entry!subentry}`, with at most two levels of sub-entries. The text of an entry can be customised: `\index{entry@\textit{entry}}`, that is, the left part of the `@` symbol is the index entry itself, used for instance for index sorting, while the right part is the text to be actually printed for the entry. Page numbers can also be formatted: `\index{entry|textit}`. Cross-referencing is also available: `\index{entry|see {other}}` and `\index{entry|seealso {other}}`. Page ranges are declared with `\index{entry|(}` and `\index{entry|)}`.

2.2 Workarounds for internationalization

Even though makeindex does not support Unicode, it can be tricked into rather properly handling Unicode index entries as shown below.

First, consider languages of the Latin alphabet but with accented letters. French is used as example here. Because makeindex does not support Unicode, it will consider bytes (i.e., ASCII characters) one by one when collecting entries and for sorting. This is problematic here since accented letters are out of sequence with other letters: in UTF-8, the three letters 'd', 'e', 'f' are each encoded with one byte, 100, 101 and 102, respectively. The accented letter 'é', which is sorted in the lexicographical order just as 'e', is encoded with two bytes: 195 and 169. Hence, from a makeindex point of view, the index

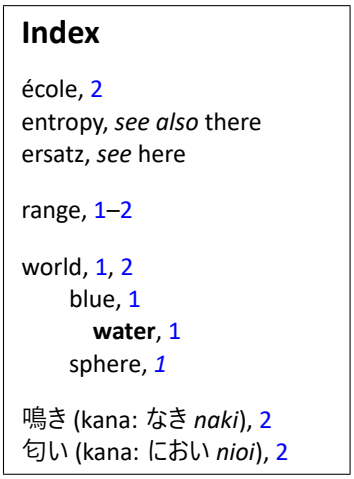


Figure 1: Sample index generated by makeindex and printed in the resulting PDF. Multilingual index entries are correctly separated and sorted, except for non-ASCII ones.

entry "école" will be sorted after "double", "entre" and "froid" where it should be "double", "école", "entre" and "froid".

This problem can be solved by using the `@` feature of index entries to provide supporting text for makeindex processing (e.g., entry sorting). Concretely, instead of declaring the entry `\index{école}`, declare `\index{ecole@école}`. In other words, accented letters are "de-accented" for index entry declaration.

Next, consider Japanese. Most characters used in Japanese are Chinese ones. These characters are only partially ordered in the Unicode standard. Moreover, this is (partial) semantic ordering, which is completely unrelated to Japanese lexicographical ordering. The latter, which is formally defined in the JIS X 4061 standard (Japanese Industrial Standards Committee, 1996), relies on the reading of a character. Such pronunciation information is represented by the small Japanese character subset kana; these characters convey no semantic information, only reading information. Fortunately (of course it was purposely done so), kana characters are represented in Unicode in their lexicographical order. For instance, the consecutive three kana characters な, に and ぬ (*na*, *ni* and *nu*, respectively) are encoded with the byte sequences 227 129 170, 227 129 171 and 227 129 172, respectively. Hence, it makes no difficulty for makeindex to sort kana index entries. Therefore, by reusing the same trick as with accented letters, it is possible to have Japanese index entries: `\index{いす@椅子}`, where the word 椅子—"chair", read *isu*—is represented by the corresponding kana characters いす.

In addition, just as with accented letters, kana characters can have variants: for instance, ば *ba* and ぱ *pa* are two variants of は *ha*. In order to correctly sort entries, such variants (here ば and ぱ) need to be replaced by the "normal" character (here は). Other variants include small characters:

for instance, ゆ for ゆ, both *yu*. More details can be found in Japanese Industrial Standards Committee (1996).

It should be noted here that the default document font is unlikely to support Japanese. Hence, after loading a new font with the `fontspec` package, say `\fntjap`, the index entry would be declared as `\index{いす@{\fntjap 椅子}}` in order to properly print the index. The index key (i.e., the left side of `@`) is only used for index entry declaration (not printing) and thus does not require the selection of an appropriate font.

The features and workarounds described in this section are illustrated in Figure 1. It can be noted that index entries are correctly separated into groups according to entries' first letters, with as a notable exception the non-ASCII entries which are improperly grouped: the Japanese entries なき and においstart with a distinct glyph and should thus be classified into two distinct groups; they are not.

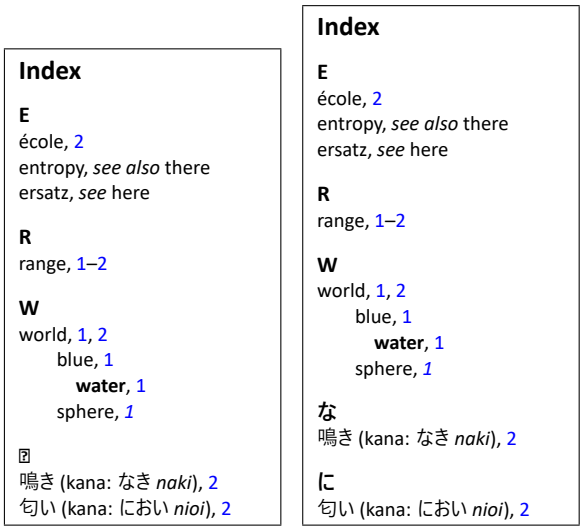


Figure 2: (a) Illustrating the makeindex issues with Unicode index entries: entry grouping and group heading problems. (b) These problems are solved by kameindex.

3 Advanced features: deriving kameindex

In addition to the briefly presented features, makeindex also enables the usage of styles for typesetting the index (Chen and Harrison, 1988; Chen, 1991). In practice, the user declares style directives inside a style file. Explicitly printing letter groups (i.e., group headings) is probably the most used style feature; it is activated with the style directive `headings_flag 1`. Such a style file is then set as second input of makeindex, in addition to the index file (the index generation flow is described below).

As mentioned earlier, when using multilingual index entries such as those of Figure 1, as makeindex is processing bytes (ASCII characters) rather than (Unicode) characters, it fails at correctly classifying entries. This issue is even more problematic when index entry groups are materialised by group headings; see Figure 2a. Again, besides the invalid group heading, the two Japanese entries should be in distinct groups.

Hence, even though workarounds exist (see Section 2), Unicode support is severely restricted with makeindex. Nonetheless, we next show that encountered issues do not prohibit completely the use of makeindex when processing Unicode documents. Index entry classification, group headings and such Unicode-related issues are addressed by kameindex. This program is used in place of the makeindex program for the index file generation process as shown in Figure 3. This figure details the document (.pdf) generation flow, starting from a LaTeX source file (.tex). The LaTeX program generates an index file (.idx) and a PDF file. (The index generation process indeed requires two passes.) Together with a style file (.ist) if any, the index file is then streamed into the makeindex (replaced by kameindex for Unicode support) program, which generates a distinct index file (.ind) for processing, beside the source file (.tex), by the LaTeX program. One should note that LaTeX is here a generic appellation which should be replaced by XeLaTeX since dealing with Unicode.

The results obtained from makeindex and kameindex for the same input can be compared in Figures 2a and 2b, respectively. It can be noted that kameindex correctly handles Unicode index entries, while still correctly handling the other ones. One important issue is font selection for non-ASCII group headings, since the group heading glyph may not be present in the default font, as with Japanese headings. In kameindex, we have implemented automatic font selection: the font selected is the one used for the first index entry of the group (or the default font if no font is specified). This is a relevant choice since the entry label and its sorting information are in the same language (unless a peculiar sorting, one that does not reflect label sorting, is specified by the user – a case non addressed in this work). In addition, as headings are often emboldened (i.e., the command `\textbf` is applied to the heading), one should not forget to declare a bold font when loading a font that does not include a bold variation for glyphs. This can be done for instance with the font loading command `\newfontfamily\fntyu{YuGothR.ttc}[BoldFont=YuGothB.ttc]` for the Microsoft Windows Japanese font "Yu Gothic". Manually specifying a bold font is usually unnecessary when loading a font by font name rather than by file name. Last but not least, one should note that, as in this font loading example, the font command name (i.e., `\fntyu`) is prefixed with `\fnt`; this is required by kameindex for font command detection.

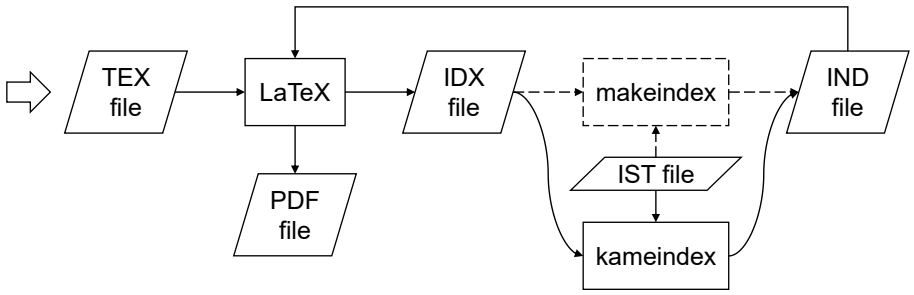


FIGURE 3: Index generation flow with kameindex instead of makeindex.

Additional non-trivial features of kameindex include: ranges, page numbers and ranges merging (e.g., a same entry containing the page numbers 1, 4, 6 and the page ranges 2–3, 5–7 will be output as the single range 1–7); subentries; cross-references; label, page number and group heading formatting; `hyperref` (RAHTZ and OBERDIEK, 2017) support (see Figure 2: page numbers are coloured to indicate links).

4 About Chinese and Korean

Chinese words are conventionally ordered alphabetically according to their pinyin transliteration (romanization). Hence, the workarounds mentioned in Section 2.2 are applicable for the indexing of Chinese words with makeindex. Concretely, index entries are declared in pinyin (non-accented) but with the entry text in Chinese characters. Also, since group headings will consist of Latin letters, they should be typeset with the default font. Thus, the font command for Chinese should not be prefixed with `fnt` so as to avoid using the Chinese font for group headings (kameindex detects font commands prefixed with `\fnt` as explained previously). For example, the index entry `\index{hanyu@{\chn 汉语}}` for the word 汉语 *hànyǔ* "Chinese [language]"; the font command here is `\chn`.

The case of Korean is more complex. The two main writing systems of Korean are hanja and hangul, with the former being based on Chinese characters. Thus, indexing of hanja entries can be done as with Chinese: romanization is used for entry sorting and grouping. As for hangul, it involves a special set of characters, with a particular ordering. First, it is important to note that hangul entries are sorted correctly by makeindex (and kameindex) since hangul glyph Unicode code points are ordered in the appropriate order. The problem which thus remains for indexing is grouping and group heading generation.

Precisely, a hangul word always starts with a consonant, called the "initial". Hence, grouping and group headings for Hangul words are necessarily based on one of the hangul initials (e.g., 'ㄱ', 'ㄴ' and 'ㄷ'). Therefore, it suffices to declare hangul index entries with the first glyph's initial as prefix. For example, `\index{ㅎ한글@{\fntkor 한글}}` for the word 한글 *hangeul* whose first glyph's (i.e., 한) initial is ㅎ. It is indeed enough to prefix the entry with the first initial, which will be used as group heading. Effectively, the entry being initially sorted correctly, adding as prefix the first initial does not impact the ordering. It is like prefixing the entries "abc", "acd" and "bde" by duplicating the first letter: "aabc", "aacd" and "bbde" retain the original entries' order.

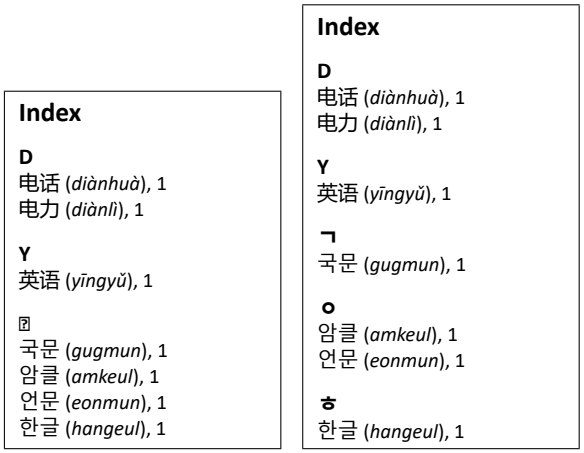


FIGURE 4: Illustrating Chinese and Korean index entries: (a) makeindex output, (b) kameindex output.

While makeindex would fail at generating properly an index for Korean entries, a sample kameindex output in the case of Chinese and Korean is given in Figure 4.

5 A more advanced example

We give in this section a more complete index example, using different alphabets. In addition to the Latin alphabet, possibly with accented letters as previously seen, Russian index entries, that use the Cyrillic alphabet, are included. When using Cyrillic letters, makeindex fails at grouping entries and creating group headers since characters are multibyte (i.e., not ASCII). Comparatively, kameindex works perfectly. As said for French, support text may be needed for Cyrillic "accented" letters as used for instance in Serbo-Croatian, Belorussian, and

Macedonian (e.g., 'Ѕ', 'Ѓ' and 'Ћ'). Greek (modern) entries, that use the Greek alphabet, are also included. Again, support text may be needed for Greek accented letters (e.g., 'ά', 'έ' and 'ή').

One should note that even though rendered similarly, some letters are distinct from a Unicode point of view. This is the case for instance with the Latin letter 'A' and the Cyrillic letter 'A'. Thus, index entries starting with such letters will be separated in two groups (i.e., Latin 'A' and Cyrillic 'A'). The same remark holds for example with the Latin letter 'E' and the Greek letter 'E' (i.e., uppercase 'ε').

An illustration of such a multilingual index is given in Figure 5; considering a same input, the makeindex and kameindex outputs are given in Figures 5a and 5b, respectively. Once again, one can see the shortcomings of makeindex. LaTeX commands for a few selected index entries are given in Table 1; it can be noticed that these are normal indexing commands, with nothing fancy. As mentioned previously, because Japanese characters are unlikely to be included in the default document font, the label (i.e., the right side of `@`) of the Japanese entry specifies the font `\fntyu`. Also, support text is provided to handle entries with accented letters and Japanese entries.

Table 1: LaTeX commands for selected index entries.

Command	Translation
`\index{юбиле́йный}`	anniversary
`\index{εψιλον@έψιλον}`	epsilon
`\index{ecole@école}`	school
`\index{ㄱ국문@{\fntkor 국문}` `(\textit{gugmun})}`	national script
`\index{なき@{\fntyu 凪}` `({\fntyu なぎ} \textit{nagi})}`	calm (wind)

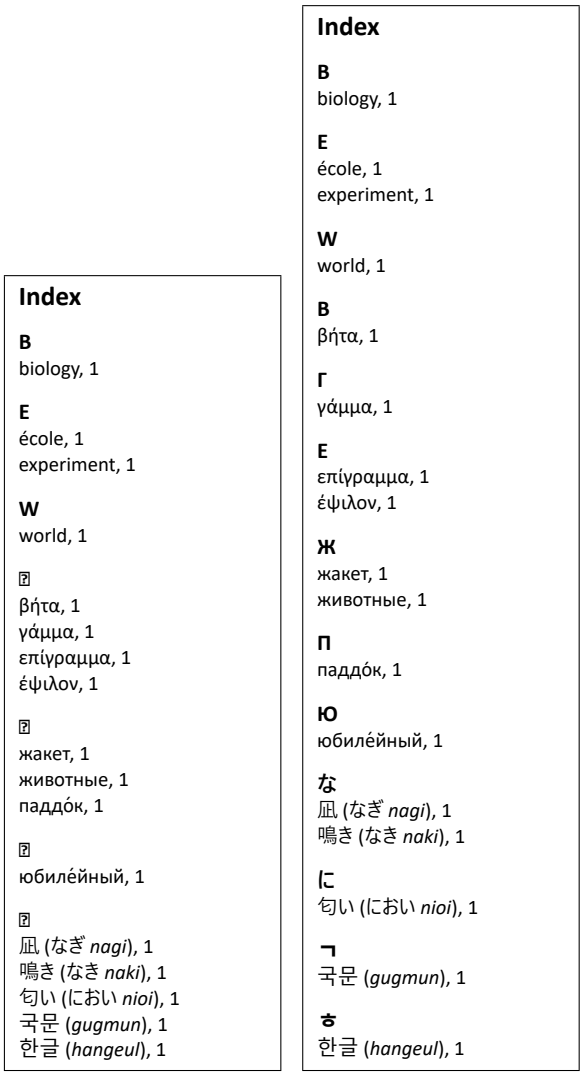


Figure 5: A more advanced multilingual index example: (a) makeindex output, (b) kameindex output.

6 Comparison and contribution

We have already shown how kameindex compares to, and supersedes, makeindex; see Section 3. In this section, we conduct additional comparison with the mainstream indexing system *texindy*, before summarising the contribution of this paper.

6.1 Comparison with related works

For multilingual documents, and Unicode in general, the indexing system *texindy* (Schrod, 2004) is recommended as a replacement of makeindex. Even though it is not our purpose with kameindex to replace *texindy*, it is worth noting the following three issues of *texindy*:

- *texindy*'s usage is comparatively complex;
- *texindy* is not compatible with `hyperref`;
- *texindy* requires manual language selection;
- *texindy* supports neither Japanese, nor Chinese, nor Korean.

All these issues are addressed by kameindex. First, the usage of kameindex is identical to that of makeindex and thus retains its usability; no additional complexity. Second, kameindex is fully compatible with `hyperref` as explained and shown in Figure 2. Third, no language selection is required with kameindex. Fourth, we have already illustrated the full Japanese support provided by kameindex.

6.2 Paper contribution

The contribution of this paper is summarised below. In addition to proposing kameindex as a Unicode-capable replacement for the makeindex program that generates index files (.ind), we have shown with concrete use cases the followings:

- When used wisely (see the discussion on workarounds in Section 2), makeindex retains a certain degree of usability with Unicode and multilingual documents. The remaining issues mostly concern the grouping of index entries,

including group heading generation, when entries are declared with non-ASCII characters.

- By replacing only a part of the makeindex subsystem (precisely, the makeindex program for index file (.ind) generation is replaced by kameindex), full Unicode support can be attained. The original makeindex package, providing for instance the `\makeindex` and `\printindex` commands, remains untouched (i.e., used as is).
- The index generation and rendering processes remain user-friendly: the only change is the call to kameindex in place of makeindex for index file (.ind) generation.

7 Conclusions

Indexing in LaTeX is conventionally realised with makeindex. However, makeindex has not been designed to support Unicode, which thus makes index creation for Unicode and multilingual documents difficult. Nevertheless, in this matter, makeindex is much more capable than often perceived. Effectively, as presented in this paper, even though designed without Unicode support, sensible usage can produce satisfactory results in a large panel of cases. The other cases are for instance the usage of index entry classification and group headers. We have proposed here the Unicode-capable kameindex program that is flow-compliant with makeindex. The usability of kameindex has been shown with concrete examples.

Regarding future works, supporting the previously described character encoding for Japanese (Bossard and Kaneko, 2018) is a meaningful objective. In order to avoid tempering with the XeLaTeX program, it would thus be necessary for kameindex to realise some character conversion when generating the index file (.ind). Also, completing the support of index style file (.ist) directives is one possible future work.

Acknowledgements

This research project is partly supported by The Telecommunications Advancement Foundation (Tokyo, Japan).

References

Bossard, A. and Kaneko, K. (2018). "Proposal of an unrestricted character encoding for Japanese". In *Proceedings of the 13th International Baltic Conference on Databases and Information Systems*. Springer, volume 838 of *Communications in Computer and Information Science*, pp. 189–201.

Chen, P. (1991). *makeindex(1): makeindex – a general purpose, formatter-independent index processor*. Unix man page. https://linux.die.net/man/1/makeindex (last accessed August 2018).

Chen, P. and Harrison, M. A. (1988). "Index preparation and processing". *Software: Practice and Experience*, **19** (9), pp. 897–915.

Japanese Industrial Standards Committee (1996). *JIS X 4061 "Collation of Japanese character string" (日本語文字列照合順番, in Japanese)*. Japanese Standards Association.

Lamport, L. (1987). *MakeIndex: an index processor for LaTeX*. Package documentation. https://ctan.org/pkg/makeindex (last accessed August 2018).

Rahtz, S. and Oberdiek, H. (2017). *Hypertext marks in LaTeX: a manual for hyperref*. Package documentation. https://ctan.org/pkg/hyperref (last accessed August 2018).

Robertson, W. and Hosny, K. (2017). *The XeTeX reference guide*. https://ctan.org/pkg/xetexref (last accessed August 2018).

Schrod, J. (2004). *texindy(1): texindy – create sorted and tagged index from raw LaTeX index*. Unix man page. http://xindy.sourceforge.net/doc/texindy-man.pdf (last accessed August 2018).

Unicode Technical Committee (2011). *The Unicode standard (version 6.0), §3.9 Unicode encoding forms D92, §3.10 Unicode encoding schemes D95*. The Unicode Consortium.

Appendix

The style directives for index generation that are supported by kameindex are listed in Table 2.

Table 2: The index style directives supported by kameindex.

Directive	Description
`headings_flag`	Activates headings if set to 1
`heading_prefix`	String used as heading prefix
`heading_suffix`	String used as heading suffix
`preamble`	String used as index prefix
`postamble`	String used as index suffix

The default values for the `preamble` and `postamble` directives are `''\\begin{theindex}\n''` and `''\n\n\\end{theindex}\n''`, respectively, purposefully matching those of makeindex. Besides, the two strings `''{\\bfseries\\large ''` and `''}\\nopagebreak\n''` are sample values for `heading_prefix` and `heading_suffix`, respectively.

▷ Antoine Bossard
Graduate School of Science
Kanagawa University
2946 Tsuchiya, Hiratsuka, Kanagawa
259-1293, Japan

▷ Keiichi Kaneko
Graduate School of Engineering
Tokyo University of Agriculture and Technology
2-24-16 Nakacho, Koganei, Tokyo 184-8588, Japan

Connecting LuaTEX to MongoDB

Roberto Giacomelli

Abstract

This paper describes in details a way to connect LuaTEX to a MongoDB database server. As a consequence, the Lua-powered typesetting engine becomes a candidate, along with other tools, to generate beautiful reports.

From local-wide project to large web applications, MongoDB—the popular NoSQL database—creates a new point of view on datasets also adopted by LuaTEX. This could extend the development perspectives.

Sommario

Questo articolo descrive in dettaglio un modo per connettere LuaTEX a un database server MongoDB, dimostrando come il motore di composizione dotato dell'interprete Lua possa candidarsi, insieme ad altri strumenti, a generare bellissimi report.

Dal piccolo progetto alle grandi applicazioni web, MongoDB — il noto database di tipo NoSQL — crea un nuovo punto di vista sui dati che si ritrova anche in LuaTEX. Ciò potrebbe incrementare ulteriormente le possibilità di sviluppo.

1 Reasons

I'm currently working on a project based on LuajitLATEX, and SQLite3[1] as a set of databases handler, to typeset a lot of high quality reports. I exposed about that in my article (Giacomelli, 2017).

LuajitTEX includes the LuaJIT FFI Foreign Function Interface, a powerful technology to bind C executable modules such as the dynamic linking SQLite3 library. Not only LuajitTEX becomes capable of collecting data by running SQL queries, but it also makes life easier in configuring the system.

Why did I discard simpler data management software such as spreadsheets or plain text files, and why didn't I ensure a full separation between TEX and data layer, for example through an agnostic file format?

First of all, the main goal of this architecture is to ensure *data safety*: that is why I chose a RDBMS system[2]. Every record saved in the archive is checked by the database engine itself that controls the integrity of the relational model, in consideration of a handful of concepts such as the primary and foreign key constraint, the `not null` clause and so on.

Secondly, the valuable advantage is that avoiding intermediate layer between data and TEX means speeding up modifications.

I have frequently adapted project entity-relationship diagrams whenever improvements were (e.g., the addition of a complex set of economic information). So, I learned a lot about how to translate real world entities into conceptual SQL models, trying to keep minimalism in mind as much as possible.

Changes were easy to apply but, after six years of development, things are no longer so easy to keep up-to-date despite the success of the project. After all, the world is constantly changing. And I consider more important finding out database systems to allow a better data representation than incrementing data bandwidth as fast as hardware potentially could do. In other words, I'm looking for more natural criteria to model a reality in rapid and unpredictable evolution.

When I talk about *project* I mean a production business where data are shared over the IT network infrastructure and automatically processed to get reports. Just think of a freelance engineer or a little design team, dealing with invoices of technical documents as part of their customer service, or a company delivering financial statements to the clients.

1.1 What I'll talk about

This paper is covering a brief description of the NoSQL unconventional way of thinking in the section 2, where I also introduce the MongoDB database system presenting fundamental concepts like the *document*.

Section 3 and the following two are dedicated to a step-by-step detailed tutorial valid for Ubuntu and Windows operating systems—section 4 and section 5 respectively. They provide instructions to set up a MongoDB local server for experimenting and a suitable MongoDB connector for LuaTEX.

Finally, from section 7 on I will discuss two demo projects simple *and* meaningful, useful to practice the projects development.

1.2 Verbatim conventions

When a shell command doesn't fit in the column width, an ending backslash \ is added to split text up into more lines.

While experimenting, you can improve readability too, splitting the lines up in your command

1. `https://www.sqlite.org`.
2. The acronym RDBMS stands for Relational Database Management System.

window according to your operating system syntax. For instance, Windows PowerShell supports Shift+Enter keys combination for multiline editing or even a backtick[3] ‘ at the end of breaking lines. Beware: PowerShell inserts a space for each newline you enter, so you won't be able to even split up a file path, while Linux or Mac OSX user can safely use backslash \ to break up to command names.

1.3 Requirements

You need a desktop computer running a 64 bit Operating System in order to execute demo projects programs illustrated from section 7 on: running locally database operations and typesetting documents for reports. A basic Lua understanding is also recommended to knowingly run such source files with LuaTEX.

2 NoSQL, Not only SQL

NoSQL philosophy denies the SQL model: no fixed schemas and no relational constraints are imposed on data. Leaving out that essential checks, web applications can take advantage of NoSQL in two main areas: a great capability to scale out—that is partitioning data across several servers, optimizing costs and performances—and a flexible design of data models. Shortly, a powerful and easy to use storage engine.

The heart of the SQL system is the *table*: a tabular dataset organized in rows and columns that describes and represents entities. Improvements require to reformulate all the dataset archived in the table. This is what we call a fixed-schema: it's just not possible to write data that don't suit the current schema.

Quite the opposite for the NoSQL databases: the information models can be eventually changed, whatever and wherever.

Google, Amazon and other Internet big companies are developing their own NoSQL data storage system. In this article, I'm going to consider the open source project MongoDB located at `www.mongodb.com`.

2.1 The MongoDB document

A MongoDB *document* is a key/value ordered list. As a matter of fact, not only the database can archive data in binary format, but can also understand the structure of the key/value data type for querying.

A document looks like this:

```
{"name":"Missouri River", "length km":3768}
```

In absence of a fixed and predefined SQL-like schema, MongoDB developers use to visualize how information is structured, simply printing documents. Thus documents literally represent themselves.

In the document above—delimited by braces—there are two fields separated by a comma. Each field has a key and a value separated by a colon. Keys are strings while values are, respectively, a string and an integer. Values are not limited to atomic types like those just considered: they can also represent compound types such as arrays—a list of comma separated values, enclosed in square brackets—or even other documents. For example, we can add geospatial information as an array of coordinates localizing the river source:

```
{
    "name" : "Missouri River",
    "length km" : 3768,
    "source coords" : [45.9275, -111.508056]
}
```

That notation—pretty much the same of the table constructor in Lua—corresponds exactly to the JavaScript object type and it highlights the strong inclination of MongoDB for web development.

2.2 Collection and database

A MongoDB *collection* is a set of documents and, in turn, a *database* is a set of collections.

Collections are referenced by name as a single UTF-8 string or as a sequence of strings concatenated with the dot character. It's the recommended way to organize document groups with namespaces.

For instance, the previous document regarding essential data about rivers could be part of the `info.basic` collection. Please pay attention to the fact that in this case there is no collection belonging to another one but it's only an optional naming rule to identify a *single* collection.

There are no constraints defined on the document structure except on the field `_id`. If the document does not have an `_id` field, MongoDB will attach one to it with a default value of type `ObjectId`, otherwise it will check if the `_id` value of any type is unique within the collection.

2.3 The JSON and BSON formats

JSON (JavaScript Object Notation) is a text-based data-interchange open standard format. It is built upon a key/value structure called *object* and a list called *array*. Values types belong to a list of seven of them like string or number, but even object and array.

The JSON specification takes only a single web page, see `https://www.json.org/`, to be completely defined. As the website says, JSON is easy for humans to read and write, and it is also easy for machines to parse and generate.

JSON is not alone in the arena of structured text formats; competitors are TOML[4], YAML[5],

3. Backtick is issued typing ALT+0096 on Windows machines.

4. Tom's Obvious, Minimal Language `https://github.com/toml-lang/toml`.

5. YAML Ain't Markup Language `http://yaml.org/`.

XML[6], RON[7] and many others, but JSON is very popular on the modern web.

The data transfer over the network is based on the serialization/deserialization process where text is encoded into an efficient binary format. BSON (Binary JSON) is the network transfer format used by MongoDB.

2.4 Working with mongo

Once MongoDB is installed, a JavaScript-based CLI client called `mongo` is available alongside the other executables.

`mongo` is a way for accomplishing administrative tasks on databases and it is a useful tool for getting started with CRUD operations and querying. The acronym CRUD stands for the four basic operations on databases: Create, Read, Update and Delete.

2.5 MongoDB learning resource

For more information on MongoDB visit the `https://docs.mongodb.com` website. It's a clear, complete and well organized documentation site. A good printed reference is the book (Chodorow, 2013).

Furthermore, online courses can be attended at `https://university.mongodb.com/` website: basic, intermediate and advanced levels can suit everybody's needs.

3 Setting up a stand-alone system

The objective is to get started with a locally-running MongoDB instance. With few adjustments you should be able to run the database server within a Local Area Network. Of course, a production-ready installation of a MongoDB server requires special treatments for security issues, an effort that goes beyond the objectives of the present work.

Anyway, a 64 bit operating system is required to run recent versions of MongoDB. No such limitation involves the client running from within a LuaTEX process.

A Lua-MongoDB connector can be build in different ways:

- a low level binding to the MongoDB C driver `http://mongoc.org/`;
- a pure Lua connector that understands the MongoDB server binary protocol over TCP network protocol;
- an intermediate node connected to LuaTEX via TCP using the library `luasocket`, that is statically linked in the Lua-powered typesetting engine. This proxy server is connected to MongoDB by means of every suitable programming languages;
- an independent execution of CLI tools like `mongo` shell itself, using files as communication channel with LuaTEX (see section 10).

Depending on the state of the art of open source projects, I chose the low level bindings. Building them requires two steps regardless of the target OS: compiling the MongoDB C driver and binding Lua to it via the LuaRocks package manager. LuaRocks must use the same Lua version of the LuaTEX interpreter, currently 5.2 for a standard TEX Live 2018 distribution.

To determine your Lua version you can read the LuaTEX manual shipped with TEX Live[8] or compile the following file with LuaTEX and check the resulting PDF:

```
% !TeX program = LuaTeX
\directlua{tex.print(lua.version)}
\bye
```

3.1 Installing a MongoDB driver for LuaTEX

The first step is to compile the MongoDB C driver and its companion. The project home is at `http://mongoc.org/`, where we can find both source code and documentation for the library `libmongoc` and the companion `libbson`. The latest library deals with the BSON format, that is the binary format used by MongoDB to store documents in the database and to perform network communication, as mentioned in the section 2.3.

The second step is to compile a Lua binding to MongoDB C driver. The project I take into account is `lua-mongo` hosted on GitHub at `https://github.com/neoxic/lua-mongo`.

In this paper I will only give detailed installation instructions for Ubuntu Linux and Windows.

4 Installing on Ubuntu

First of all it's important to notice that the MongoDB binary for Ubuntu has been compiled with SSL enabled and dynamically linked. This requires the SSL libraries to be seperately installed on your system with the following shell command:

```
$ sudo apt-get install libcurl4 openssl
```

Adding to the repository list the official MongoDB software archive, you can also install the binary via the standard `apt` package manager. Doing so the package will be updated automatically. However, the manual procedure allows you to keep things under a complete control.

4.1 Installing MongoDB

MongoDB installation is very simple because you only have to download and unzip a file: select

6. Extensible Markup Language `https://www.w3.org/XML/`.
7. Rusty Object Notation `https://github.com/ron-rs/ron`.
8. To access your TEX-related documentation just type `texdoc` ⟨*package*⟩ in a terminal session. For instance, `texdoc luatex`.

Table 1: Direct links to the downloadable files of the MongoDB Community Server package available at the time of writing for the main 64 bit desktop OS.

OS identifier/Direct download link
Ubuntu 18.04 LTS 64 bit https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-ubuntu1804-4.0.1.tgz
Windows 64 bit installer https://fastdl.mongodb.org/win32/mongodb-win32-x86_64-2008plus-ssl-4.0.1-signed.msi
OSX 64 bit https://fastdl.mongodb.org/osx/mongodb-osx-ssl-x86_64-4.0.1.tgz

the Community Server tab from the official MongoDB download web page https://www.mongodb.com/download-center and pick your choice selecting your preferred Operating System. You can also grab your installation package from the direct link shown in table 1.

Placing executables in a system directory is safer due to the superuser rights protection, but for testing operations I prefer to temporarily keep the binary as well as the database files in my home directory:

```
$ tar -zxvf \
  mongodb-linux-x86_64-ubuntu1804-4.0.1.tgz
$ cp -r \
  ./mongodb-linux-x86_64-ubuntu1804-4.0.1 \
  ~/mongodb
$ mkdir ~/mongodb/data
```

The following commands make the server start on the localhost network address with no user authentication—in production environments a safe access to the server must be configured:

```
$ cd ~/mongodb/bin
$ ./mongod --smallfiles --dbpath ~/mongodb/data
```

Carefully look at the output produced by `mongod`: you can check if the system is up and correctly running. To safely stop the server press CTRL+C.

It's also recommended to install a copy of the Compass Community Edition GUI client, looking for related tab form in the download MongoDB web site. In figure 1 is shown a screenshot of Compass.

4.2 Compiling MongoDB C Driver

As a preliminary step, we need to install some packages from the Ubuntu repository. These shell commands do the job:

```
$ sudo apt-get install build-essential cmake
$ sudo apt-get install liblua5.2-dev
$ sudo apt-get install libssl-dev libsasl2-dev
```

As reported in the installation guide of the MongoDB C driver, download, unzip and prepare the project with:

```
$ wget \
  https://github.com/mongodb/mongo-c-driver\
  /releases/download/1.12.0\
  /mongo-c-driver-1.12.0.tar.gz
$ tar xzf mongo-c-driver-1.12.0.tar.gz
$ cd mongo-c-driver-1.12.0
$ mkdir cmake-build
$ cd cmake-build
$ cmake \
  -DENABLE_AUTOMATIC_INIT_AND_CLEANUP=OFF ..
```

If everything is OK, you may go on and install the driver—make sure that the working current directory is `cmake-build`—:

```
$ make
$ sudo make install
```

4.3 Installing LuaRocks

LuaRocks instruction from its website https://luarocks.org/ are quite clear:

```
$ wget https://luarocks.org/releases\
  /luarocks-3.0.0.tar.gz
$ tar zxpf luarocks-3.0.0.tar.gz
$ cd luarocks-3.0.0
$ ./configure; sudo make bootstrap
```

4.4 Installing lua-mongo bindings

The very last shell command is:

```
$ sudo luarocks install lua-mongo
```

4.5 Post-install adjustment

To provide *visibility* to the Lua MongoDB binding from within LuaTEX, the library file could be copied in the `bin` directory of the main TeX Live tree, accordingly to the system variable `$CLUAINPUTS`. In fact, it contains paths where the typesetting engines looks for the C module.

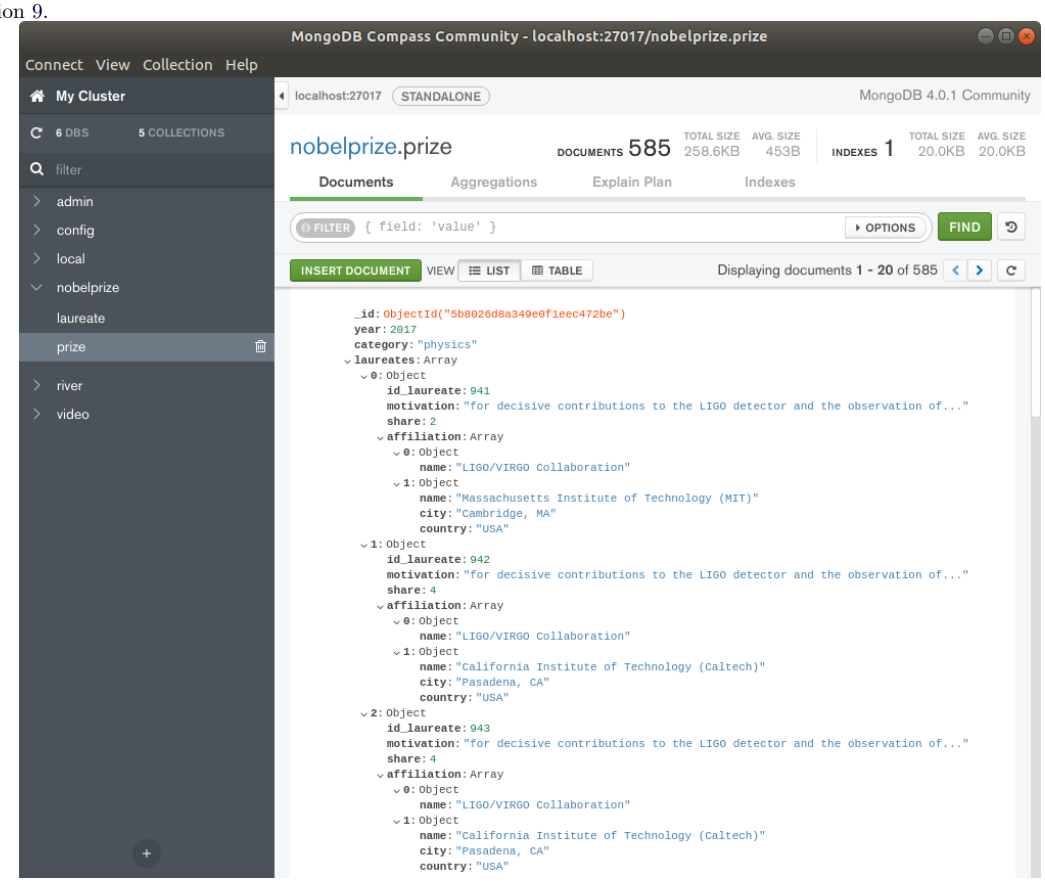
Indeed the LuaTEX manual reports the variable value as the following pattern:

```
CLUAINPUTS= .:\
  $SELFAUTOLOC/lib/{$progname,$engine,}/lua//
```

The first directory in the pattern is simply the folder containing the `.tex` source file itself, identified by the dot char '`.`', but what about the next `$SELFAUTOLOC` variable? To find out the answer we can address LuaTEX itself, compiling the source file below:[9]

9. We obtain the same answer if we invoke the command `kpsewhich -var-value= $SELFAUTOLOC` from the terminal.

FIGURE 1: A screenshot of Compass Community Edition GUI client running on Ubuntu 18.04 while connected to the local MongoDB server. It shows a document of the `prize` collection belonging to the `nobelprize` database, part of the project defined in section 9.




```
% !TeX program = LuaTeX
\directlua{
  print(kpse.expand_var [[$SELFAUTOLOC]])
}
\bye
```

that prints on the standard output the variable expansion as:

```
/usr/local/texlive/2018/bin/x86_64-linux
```

Essentially, C modules home can be only a platform-specific directory in the main TDS tree[10], and this in far from the ideal solution because the main tree is something that users should leave untouched, giving out local or personal tree of distribution to settle what is not officially part of TEX Live.

A workaround may be directly using the loader internally called by the Lua function `require()`, that is the standard way to load a module. `require()` has a default list of search paths included those derived from the pattern in the `$CLUAINPUTS` variable.

While we will discuss this in the section 6, let's now complete the installation: we only have to copy the C module in the local tree of TEX Live with the command:

10. For more information on TDS tree structure run the shell command `texdoc tds`.

```
$ cd /usr/local/texlive/texmf-local
$ sudo mkdir -p ./scripts/mongo
$ sudo cp /usr/local/lib/lua/5.2/mongo.so \
  ./scripts/mongo/
```

5 Installing on Windows

Since this task is slightly long, you may find comfortable to help yourself some coffee or tea, cookies and so on, and relax. Let's start installing Visual Studio Community latest release, currently 2017, from Microsoft website `https://visualstudio.microsoft.com/en/downloads/`. If necessary, register your Visual Studio copy submitting an account from the dedicated command you find in the Help menu.

Later, it will be useful Microsoft Build Tools 2015 too, so install it from Microsoft Download Center but only *after* installing Visual Studio.

You also need to install `cmake` from `https://cmake.org/download/`. You should have a 64 bit OS so choose win64-x64 architecture and not win32-x86 one.

5.1 Installing MongoDB

The Windows installer—the `.msi` file reported in the table 1—gives you two different kinds of configuration: running MongoDB as a service or not. The second option is recommended for testing and local

project according to our experimental purpose. In any case, you may wish to install the Compass utility too, a GUI client for MongoDB databases. If so, check on the related flag in the Setup Wizard and press the Next button to enrich the final step.

To locally start the server, run in a console window the following commands:

```
> mkdir C:\mongodb-data
> cd 'C:\Program Files\MongoDB\Server\4.0\bin\'
> .\mongod --smallfiles --dbpath C:\mongodb-data
```

Check the output to assess if all is OK, then stop the server pressing CTRL+C to send a shutdown signal to the process.

5.2 Compiling MongoDB C Driver

Download the source code of the latest version of the C driver from http://mongoc.org/#download, current release is 1.12.0, unzip the file mongo-c-driver-1.12.0.tar.gz, then open a console on the uncompressed folder.

According to the documentation, the commands to create a Visual Studio 2017 project are the following (the double dot in the last line is important):

```
> cd mongo-c-driver-1.12.0
> mkdir cmake-build
> cd cmake-build
> cmake -G "Visual Studio 15 2017" \
  "-DCMAKE_INSTALL_PREFIX=C:\mongo-c-driver" \
  "-DCMAKE_PREFIX_PATH=C:\mongo-c-driver" \
  ..
```

If you don't know how to split the last command into several lines inside the shell, simply type it in a single line.

Since the luatex executable is a 32 bit x86 binary, make sure Visual Studio is set for Win32 platforms before compiling. To inspect which architecture luatex is compiled for, digit the shell command:

```
> luatex --version
```

If the answer is (or looks like)

```
This is LuaTeX, Version 1.07.0 \
  (TeX Live 2018/W32TeX)
```

it's evident that we have a Win32 compliant program.

Back to the source folder, open the generated file mongo-c-driver.sln in Visual Studio, select into the explorer the solution named ALL_BUILD, right click on it and run the command *Compile*. When the process ends, repeat the same steps with the solution named INSTALL. At this point you should see in C:\mongo-c-driver directory the files you were waiting for.

Add to the system PATH variable the folder C:\mongo-c-driver\bin. This allows Windows to find the binary. In order to easily open the dialog, type in the search bar "system environment variable" and select the proper icon.

5.3 Installing Lua Libraries

Pursuing a Lua driver installation for MongoDB, make sure you have Lua itself installed with the same version of LuaTEX, as explained before. If not, it is very simple, thanks to the project Lua Binaries, getting that files.

Browse the web page http://luabinaries.sourceforge.net/ and pick up Lua 5.2.4 release pages hosted on SourceForge. Click in sequence on the folder Windows, hence Dynamic, and download the file lua-5.2.4_Win32_dll15_lib.zip. The file name tells whether the platform we have chosen is correct: Win32 shows that library is for 32 bit binary and dll15 is about the latest Visual Studio version.

Please make sure you have clicked on the Dynamic and not on the Static link or you will suffer soon or later the error *Multiple Lua VM's detected*. As a comparison, in the file name you must read the string dll, standing for dynamic linking library in the Microsoft ecosystem terminology.

Unpack the files and copy all of them in the folder, say C:\Lua, as a name easy to trace and remember. That folder will be the home of LuaRocks too, the next main character of our "play".

From the same repository, download a Lua interpreter clicking on the link folder Tools Executables. The file should be lua-5.2.4_Win32_bin.zip. Unzip it in the same directory where you are going to place the libraries file.

Add to the system PATH variable the folder C:\Lua, then run the interpreter.

5.4 Installing LuaRocks

The starting point for the Lua most famous package manager is the official website https://luarocks.org/. The latest release is 3.0.1 but at the moment it is unable to complete the installation process due to a dynamic linking failure. Fall back on 2.4.4 version downloading the file luarocks-2.4.4-win32.zip from http://luarocks.github.io/luarocks/releases/.

Unzip it and run this command:

```
> .\install.bat /P 'C:\Lua\Luarocks244' \
  /CONFIG 'C:\Lua\Luarocks244'
```

The option /P gives the location where LuaRocks will be installed, while /CONFIG gives the location of the configuration file.

As it should be clear now, add to the user variable the entries LUA_PATH and LUA_CPATH as follow (type the first, that has two paths, in one line):

```
LUA_PATH = C:\Lua\systree\share\lua\5.2\?.lua;
C:\Lua\systree\share\lua\5.2\?\init.lua

LUA_CPATH = C:\Lua\systree\lib\lua\5.2\?.dll
```

We have to append the path of the MongoDB C driver binaries

```
external_deps_dirs = {
    "c:/mongo-c-driver",
}
```

in the LuaRocks configuration file `config-5.2.lua`.

5.5 Installing the `lua-mongo` binding

Navigate to the folder `C:\Program Files (x86) Microsoft Visual C++ Build Tools`. Right click on "Visual C++ 2015 x86 Native Build Tools Command Prompt", and select "Run as administrator", then in the opened console window, run the following command:

```
"C:\Program Files (x86)\LuaRocks\luarocks.bat" \
  install lua-mongo
```

In order to allow LuaTEX to load the driver using the strategy explained in section 4.5, as the very last commands run the following sequence:

```
> cd C:\texlive/texmf-local
> mkdir scripts
> cd scripts
> mkdir mongo
> cd mongo
> copy "C:\Lua\systree\lib\lua\5.2\mongo.dll"
```

6 Testing

In the end, the connector binary file is in the local tree, more specifically, in the sub-directory `$TEXMFLOCAL/scripts/mongo`. It can't be loaded by the usual Lua function due to the restricted list of search path set for `require()`, as explained in the section 4.5.

The Lua function `package.loadlib` fits our case. It returns the initialization function of a C module written according to the Lua C API, that returns the actual library. The expected arguments are the path to the file and the name of the initialization function beginning with `luaopen_`.

Instead of hard-coding the library path, a more general option is to call the LuaTEX specific function `expand_var()` from the `kpathsea` utility module; in this way we get the value of the `$TEXMFLOCAL` variable and take into account the file extension depending on the Operating System—`.dll` for Windows, `.so` for other OS—calling the function `os.type()`.

The code that implements such solution is the following:

```
% !TeX program = LuaTeX
\directlua{
  local path = kpse.expand_var [[$TEXMFLOCAL]]
  assert(path)
  if os.type == [[windows]] then
      path = path..[[/scripts/mongo/mongo.dll]]
  else
      path = path..[[/scripts/mongo/mongo.so]]
  end
  local _mongo = package.loadlib(
      path,
      "luaopen_mongo"
    )
  assert(_mongo)
  local mongo = assert(_mongo())
  % print the available functions
  local par = string.char(92)..[[par]]
  for k, v in pairs(mongo) do
    if type(v) == [[function]] then
      tex.print(k..[[()]]..par)
    end
  end
}
\bye
```

As a general test, that `.tex` file prints also in the PDF file the list of the functions available in the connector `lua-mongo`. If the compilation process ends correctly, the minimal test can be considered passed and the functions list appearing regardless of the order should be the following:

```
Double()
Int64()
Decimal128()
Regex()
BSON()
Timestamp()
Binary()
DateTime()
ObjectID()
Int32()
ReadPrefs()
type()
Client()
Javascript()
```

That list can be compared with the API documentation of the project at the URL `https://github.com/neoxic/lua-mongo/blob/master/doc/main.md`.

7 LuaTEX application

In the next two sections I present example projects involving operations on a MongoDB database and queries execution from within LuaTEX in order to typeset reports.

Be sure you have a `mongod` server instance locally up and running while you compile with LuaTEX. So, along with a modern TEX distribution, you have installed several components in order to compile the source code examples. In case you didn't do that yet, please head to section 3 for details.

All of the project files are downloadable from ArsTEXnica home page at `https://www.guitex.org/home/it/arstexnica`. Browse issues online page to find out the link pointing to the related compressed archive named `mongodb-project.zip`.

8 Rivers

The first example models a very simple dataset regarding the longest rivers in the United States[11], including a one-to-many relationship: alongside basic information like river name and length, there is the list of the crossed states too.

The dataset would be represented in a SQL schema with three different tables:

- table `river` with the basic information;
- table `regions` defining states names;
- a pure relational table `crossing` connecting rivers with states through a pair of foreign key identifiers.

On the contrary, thanks to the recursive nature and compound data types of MongoDB documents, we can represent rivers and crossed regions by them with only one document, within a single collection.

A document may contain the list of the rivers regions as an array assigned to the field `regions` as the following:

```
{
    ”name” : ”Missouri River”,
    ”length_km” : 3768,
    ”regions” : [
        ”Montana”,
        ”North Dakota”,
        ”South Dakota”,
        ”Nebraska”,
        ”Iowa”,
        ”Kansas”,
        ”Missouri”
    ]
}
```

We can even enrich the previous model because the regions may be represented as an array of embedded documents as showed in the listing below:

```
{ // embedded document version
    ”name” : ”Missouri River”,
    ”length_km” : 3768,
    ”regions” : [
        {
            ”state”:”Montana”,
            ”code”:”MT”,
            ”shortname”:”Mont.”
        },
        {
            ”state”:”North Dakota”,
            ”code” :”ND”,
            ”shortname”:”N. Dak.”
        },
        {
            ”state”:”South Dakota”,
            ”code” :”SD”,
            ”shortname”:”S. Dak.”
        },
        {
            ”state”:”Nebraska”,
            ”code” :”NE”,
            ”shortname”:”Nebr.”
        },
        {
            ”state”:”Iowa”,
            ”code” :”IA”,
            ”shortname”:”Iowa”
        },
        {
            ”state”:”Kansas”,
            ”code” :”KS”,
            ”shortname”:”Kans.”
        },
        {
            ”state”:”Missouri,
            ”code” :”MO”,
            ”shortname”:”Mo.”
        }
    ]
}
```

In MongoDB we may decide to *embed* or to *reference* data. While the first technique is represented by the latest river model, the second consists in just including the reference to the region documents instead of the documents themselves. Those references have the form of `_id` values of the corresponding documents in a different and unspecified collection.

Deciding which is the best solution depends on the performance of the application context and not on the normalization criteria that leads the design of SQL schemas.

8.1 Making the database

In order to create the database, the simplest method is to write a JavaScript file `river.js` to be executed in the `mongo` shell with the command:

```
$ mongo --host localhost river.js
```

The code must define names, collection, and rivers documents, structured as formerly seen in the first model for the longest river in the USA, where regions are a simple list of strings. The script `river.js` can be illustrated by the listing below where a three dots sequence shortens lines but not the essence:

```
// get/define database object
db = db.getSiblingDB(”river”);

// data insertion
db.generalinfo.insertMany([
    {...}, {...}, {...}, ...
])
```

The pre-instantiated `db` variable refers to the current database. The method `getSiblingDB()` of the db type returns a pointer the requested database which will be created if it does not exist. The next expression is a chain where in the first stage the

11. Data source https://en.wikipedia.org/wiki/List_of_longest_rivers_of_the_United_States_(by_main_stem).

collection `generalinfo` of the database pointed by db is returned—once again, if the collection doesn't exist it will be created—then its method `insertMany()` will write an array of documents.

Too many words to explain such a few lines of smart and practical code. `mongo` shell is inspired by SQL CLI clients like `psql` in PostgreSQL, but its JavaScript library is designed with the following motto in mind: *learn and try* then *configure*.

8.2 Querying with `mongo` shell

It is a very common way in MongoDB that methods performing operations on a database take as an argument a document object. For instance, in `mongo` shell you can retrieve rivers that are exactly 2000 km long by the following query:

```
> use river
switched to db river
> db.generalinfo.find({"length_km": 2000})
```

Querying that on our dataset we obtain the following document:

```
{
   "_id" : ObjectId("5b7af5a589923a94e7ce4cb0"),
   "name" : "Columbia River",
   "length_km" : 2000,
   "regions" : [
       "British Columbia",
       "Washington",
       "Oregon"
   ]
}
```

As expected, MongoDB has automatically added the mandatory `_id` field to each document inserted by the `river.js` script.

8.3 This looks like a job for LuaLaTeX

Our goal is to typeset the tabular shown in table 2. This is what we have to do:

1. create a client that has established a connection with the MongoDB server;
2. get the collection object `generalinfo` of the `river` database;
3. perform the query to retrieve all the rivers information;
4. typeset the table with a `tabular` environment.

All these steps are straightforward. To keep the `.tex` source code simple and a little bit more abstract, I'm going to write an auxiliary Lua library in the external file `libmongo.lua` to be located in the same directory of the main source file.

The constructor `Connect()` takes as an argument the database name, then establishes a client connection to the MongoDB server on `localhost`.

The real job is demanded to the Lua binding of MongoDB C Driver, compiled as described in sections 4 (for Ubuntu)–5 (for Windows). The loading mechanism of the binding library is explained in section 6.

No further functions but the method `FindIn()` are required. In the previous task list, it performs the 2nd and 3rd tasks and the Lua array with all the documents selected by the query is returned.

The `libmongo.lua` code is the following:

```
local lib = {}
lib.__index = lib
-- constructor
function lib:Connect(dbname)
    assert(type(dbname)=="string")
    local path = kpse.expand_var [[$TEXMFLOCAL]]
    assert(path)
    if os.type == [[windows]] then
       path = path..[[/scripts/mongo/mongo.dll]]
    else
       path = path..[[/scripts/mongo/mongo.so]]
    end
    local _mongo = package.loadlib(
        path,"luaopen_mongo"
    )
    assert(_mongo)
    local mongo = assert(_mongo())
    local client = assert(
        mongo.Client("mongodb://127.0.0.1")
    )
    local o = {
        _mongo  = mongo,
        _client = client,
        _dbname = dbname,
    }
    setmetatable(o, self)
    return o
end
-- query method
function lib:FindIn(
    collection, query, option, prefs
    )
    assert(type(collection) == "string")
    local client = self._client
    local coll = client:getCollection(
        self._dbname, collection
    )
    local data = {}
    query = query or {}
    for doc in coll:find(query, option, prefs)
                   :iterator() do
        data[#data + 1] = doc
    end
    return data
end

return lib
```

According to the object oriented paradigm in Lua, we have to use the *colon notation* and not the dot operator to call methods. If these sentence sounds weird to you, please refer to the book (Ierusalimschy, 2016) and read the chapter dedicated to metamethods and metatables and the chapter dedicated to Object Oriented Programming.

Table 2: The longest United States of America's rivers as reported by Wikipedia. The table is typeset by LuaTEX directly connecting to a MongoDB database as explained step by step in the text.

River name	Length (km)	Regions
Missouri River	3768	Montana, North Dakota, South Dakota, Nebraska, Iowa, Kansas, Missouri
Mississippi River	3544	Minnesota, Wisconsin, Iowa, Illinois, Missouri, Kentucky, Tennessee, Arkansas, Mississippi, Louisiana
Yukon River	3185	British Columbia, Yukon Territory, Alaska
Rio Grande	2830	Colorado, New Mexico, Texas, Chihuahua, Coahuila, Nuevo León, Tamaulipas
Colorado River	2330	Colorado, Utah, Arizona, Nevada, California, Sonora, Baja California
Arkansas River	2322	Colorado, Kansas, Oklahoma, Arkansas
Columbia River	2000	British Columbia, Washington, Oregon
Red River	1811	Oklahoma, Texas, Arkansas, Louisiana
Snake River	1674	Wyoming, Idaho, Oregon, Washington
Ohio River	1575	Pennsylvania, Ohio, West Virginia, Indiana, Illinois, Kentucky

The last task—number 4 in the list—is reserved to the LuaLaTEX source file itself:

```
% !TeX program = LuaLaTeX
\documentclass{standalone}
\usepackage{fontspec}
\defaultfontfeatures{Ligatures=TeX}
\setmainfont{CMU Serif}
\usepackage{booktabs}

\directlua{
    local libmongo = require "libmongo"
    local db = libmongo:Connect("river")
    river = db:FindIn("generalinfo")
}
\begin{document}
\begin{tabular}{lcp{120mm}}
\toprule
River name & Length (km) & Regions\\
\midrule
    \directlua{
    local stop = string.char(92)
    stop = stop..stop
    for _, r in ipairs(river) do
        tex.print(r.name)
        tex.print("&")
        tex.print(r.length_km)
        tex.print("&")
        tex.print(table.concat(r.regions, ", "))
        tex.print(stop)
    end
    }
\bottomrule
\end{tabular}
\end{document}
```

Reading this code, two elements worth a comment: first of all, the field `regions` are an array of strings as in the documents, so the Lua binding uses the Lua table type as the direct incarnation of the MongoDB document: in fact, the variable `r` in every cycle has the keys `name` and `length_km` as well as the key `regions` that refers to a further Lua table in the array form. The Lua standard function `table.concat()` provides the string corresponding to the comma separated values of States and provinces crossed by river.

As the second interesting point, the query returns documents in the same order of the original insertion, which is the order of the Wikipedia table at the time I wrote the `river.js` script.

An optional parameter allows us to explicitly set the order, sorting documents by value. Try to edit the previous code as in:

```
\directlua{
    local libmongo = require "libmongo"
    local db = libmongo:Connect("river")
    river = db:FindIn(
        "generalinfo",
        {},
        {sort = {length_km = 1}}
    )
}
```

The integer 1 specifies ascending order while -1 specifies descending order, in our case applied to the rivers length.

Furthermore, special query document fields named *operators* act as conditional parameters being part of the MongoDB query language. They have a dollar sign as their first character, like in `$gte`—that stands for *greater or equal* or $\geq$. For instance, in the previous code we can select only those rivers that have a length greater or equal to 2000 km, adding a specific query document as the second argument of the `FindIn()` method:

```
river = db:FindIn("generalinfo",
    { length_km = { ["$gte"] = 2000 }}
)
```

or even by a length range like in:

```
river = db:FindIn("generalinfo",
   length_km = {["$gte"] = 2000, ["$lte"]=3000}}
)
```

We can execute the same query in the `mongo` shell as in the following session:

```
> use river
switched to db river
> db.generalinfo.find({"length_km":
... {"$gte":2000, "$lte":3000}
... })
```

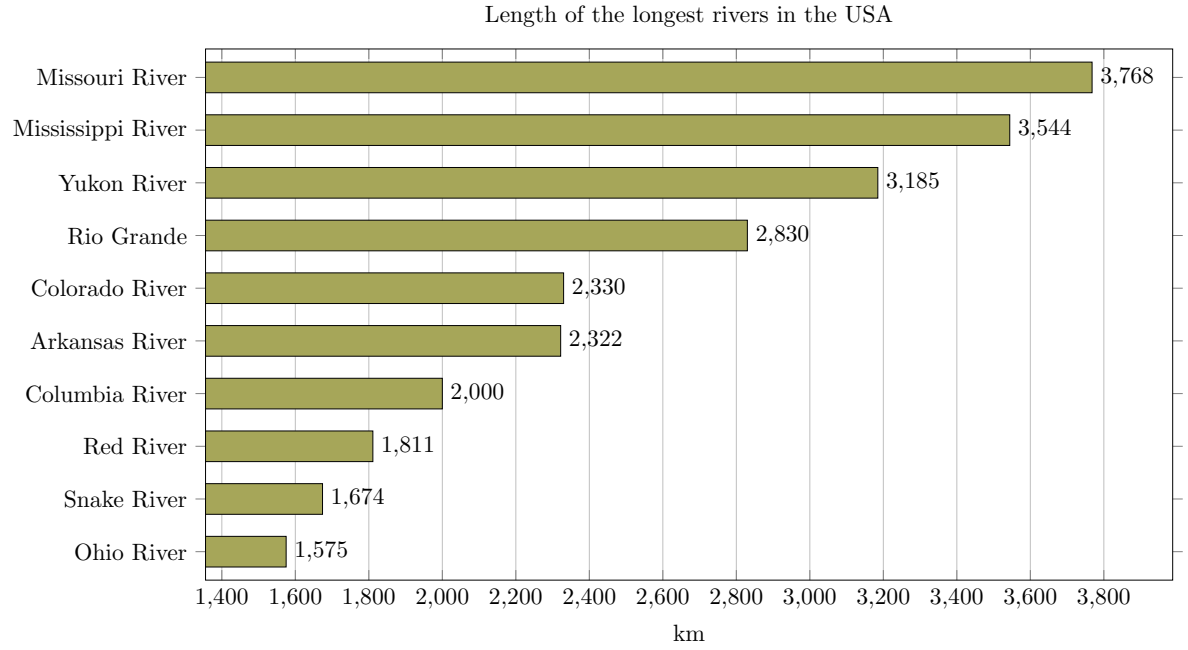


FIGURE 2: The longest United States of America's rivers as reported by Wikipedia. The histogram plot is typeset by LuaTEX performing a direct connection to a MongoDB database as explained step by step in the text.

8.4 Histogram

Once data is retrieved from MongoDB and stored in Lua tables, LuaTEX can typesets information in whatever form the user requires, e.g., a histogram plot. The figure 2 is the PDF output of the following source file where pgfplots package plots rivers length:

```
% !TeX program = LuaLaTeX

\documentclass[margin=2pt]{standalone}
\usepackage{fontspec}
\defaultfontfeatures{Ligatures=TeX}
\setmainfont{CMU Serif}

\usepackage{pgfplots}
\pgfplotsset{compat=1.16}

\directlua{
    local libmongo = require "libmongo"
    local db = libmongo:Connect("river")
    river = db:FindIn("generalinfo", {},
        {sort={length_km = 1}}
    )
}
\newcommand\coords{\directlua{
    for _, riv in ipairs(river) do
    tex.print(
        "(".."..riv.length_km..","..riv.name..")"
    )
    end
}}
\newcommand\riverlist{\directlua{
    local t = {}
    for i, riv in ipairs(river) do
        t[i] = riv.name
    end
    tex.print(table.concat(t, ","))
}}

\begin{document}
\begin{tikzpicture}
\begin{axis}[
    xbar,
    bar width=13pt,
    width=16cm,
    height=9.5cm,
    enlarge y limits=0.060,
    xmajorgrids,
    title={%
    Length of the longest rivers in the USA},
    xlabel={km},
    symbolic y coords/.expanded={\riverlist},
    ytick=data,
    nodes near coords,
]
\addplot[draw=black,fill=blue!35!yellow]
    coordinates {\coords};

\end{axis}
\end{tikzpicture}
\end{document}
```

9 Nobel Prize

Appending to the URL http://api.nobelprize.org/v1/ one of the strings laureate.json, prize.json or country.json, everyone can download data regarding Nobel Prizes in JSON format.

For more information about Nobel Prize API please visit the website https://www.nobelprize.org/about/developer-zone-2/.

This dataset, freely available from the Nobel Foundation, contains many-to-many relationships,

such as people who won more than one Nobel Prize and single Nobel Prizes won by more than one person.

That problem is complex enough to deserve an interesting project to be developped in MongoDB: how should we define data models for such an admirable dataset?

9.1 Modelling documents

Instead of embedding data in a single structured document, we will consider a pair of distinct models representing *laureate* and *prize*.

A listing is worth a thousand words, so let start with a possible laureate document:

```
{ // basic model for laureate
    "_id": 1,
    "firstname": "Wilhelm Conrad",
    "surname": "Röntgen",
    "born": new Date("1845-03-27"),
    "died": new Date("1923-02-10"),
    "bornCountry": "Prussia (now Germany)",
    "bornCountryCode": "DE",
    "bornCity": "Lennep (now Remscheid)",
    "diedCountry": "Germany",
    "diedCountryCode": "DE",
    "diedCity": "Munich",
    "gender": "male"
}
```

The fields `born` and `died` have type `Date`—they are instantiated objects—and the remaining fields, but the document identifier `_id` which is an integer, have type `string`.

A very simple model after all, and also flexible: for instance, if the laureate is alive, the field `died` does not exist as well as `diedCountry`. Furthermore, laureate can be an organization without any first name.

A more structured model could represent the prize:

```
{ // basic model for prize
    "year": 1901,
    "category": "physics",
    "laureates": [
      {
        "id_laureate": 1,
        "motivation": "in recognition of ...",
        "share": 1,
        "affiliation": [
          {
            "name": "Munich University",
            "city": "Munich",
            "country": "Germany"
          }
        ]
      }
    ]
}
```

The field `laureates` is an array of documents, each one representing a winner referenced through the field `id_laureate`. The field `affiliation`—one more time—is an array because scientists can have more than one affiliation, for instance when they are part of world-wide research program in addition to their Academic Institution.

But why affiliations fields aren't in the laureate model? After all Nobel Prize is unrelated to the institution where laureates carry out their studies.

This is right but what about the case of a laureate who won more than one prize, each one being afferent to different institutes or organizations? It is a very uncommon event, but we can't forget the case of Marie Curie.

In fact, affiliation is something that may change during life, so we have to trace such a piece of information while the Nobel Prize is awarded.

The introduced *basic model* fulfills that requirementin a practical way in spite of coherence. A different possible way is to reference a third referenced document `affiliation` that references `laureate` such as:

```
{ // alternative prize model
    "year": 1901,
    "category": "physics",
    "laureates": [
      {
        "id_affiliation": 100, // reference
        "motivation": "in recognition of ...",
        "share": 1,
      }
    ]
}
```

and

```
{ // alternative affiliation model
    "_id": 100,
    "id_laureate": 1,
    "affiliation": [
        {
          "name": "Munich University",
          "city": "Munich",
          "country": "Germany"
        }
    ]
}
```

while the laureate document remains the same.

These alternative models require at least three queries to know who won a specific prize: the first one to retrieve the affiliation identifier, another one to query the affiliation document to get the laureate identifier, and finally to query the laureate document. Vice versa the same task with the basic model takes only one query for the prize document and one for laureate document.

Talking about the alternative three-parted model, it is rather interesting the document reference scheme for people who won the prize twice: there will be two prizes referencing two different affiliations, each one referencing the same laureate: a link figure similar to an upper case V.

From now on I will adopt the basic model counting the collections `laureate` and `prize` within the

nobelprize database. It is out of the scope of this work an in-deep exploration of design pattern for MongoDB applications. Nevertheless, it is important to show an example of document referencing as counterpart of document embedding. Further information on design pattern application in MongoDB can be found in the book (Copeland, 2013) from O'Reilly Media.

MongoDB doesn't offer fast join functions between tables as in SQL. From the server point of view referencing documents means a larger number of queries when data are required, while embedding document means a larger number of updating when data are changed. Pros and cons assessment is focused on server workload rather than data coherence, as in the case of high volume website or big data archive.

9.2 Data validation

What I didn't know about the Nobel Prize was the meaning of the field share in the prize document. If the Nobel Prize is won by more than one laureate, they share the prize in proportion and not always in equal parts. As an example, if three laureates share the prize respectively with 2, 4, 4 quote, the first one deserves one half of the prize while the others deserve a quarter each.

I'm not digressing. It should be true that the sum of the inverse of each prize share values is exactly equal to 1. Prior MongoDB 3.2 version, client-side validation was the only option to find out errors that MongoDB wouldn't have discovered in absence of any server-side constraints usually active in a SQL schema.

Recent versions of MongoDB are able to validate documents before update or insert operations concerning a single collection. For instance, we can validate every values types, the existence of a mandatory field and the binding of a value to an enumeration.

As an example on such server-side validation, we can explicitly create the laureate collection adding a validator as an object with the $jsonSchema operator. The following JavaScript code shows an example:

```
db.createCollection("laureate", {
   validator: {
      $jsonSchema: {
         bsonType: "object",
         required: [ "surname", "gender" ],
         properties: {
            firstname: {
               bsonType: "string",
               description: "must be a string"
            },
            surname: {
               bsonType: "string",
               description: "must be a string"
            },
            gender: {
               enum: ["female", "male", "org"],
               description: "enum values only"
            },
         }
      }
   }
})
```

Validator may be also a query filter expression, generally less expressive than a $jsonSchema operator.

Server-side validation is only a condition, that can be true or false, about acceptability of documents whereas client can even edit a non-compliant document according to specific rules, instead of rejecting such objects.

In my opinion, in most cases it is desirable to add a server-side validator to every collection. Anyway in the next section there will be an example of client-side validation, mainly to provide a very helpful method on how to execute JavaScript code.

9.3 A Method for client-side data validation

The basic model defined in the previous section requires to apply several adjustments to the original JSON file spread by Nobel Prize Foundation. With a mix of regular expressions and JavaScript code, I have translated the original files into the structure required by the basic model. The final result consists of two JavaScript scripts called laureates.js and prizes.js.

Each script contains only one assignment: a variable takes a literal array of objects. I'm going to show how to load these files very soon. In a compact syntax the laureates script may be written as:

```
// laureates.js
var laureate = [{...}, {...}, ...];
```

Now suppose such variables—laureate and prize—are available in a JavaScript environment: we could check everything about dataset integrity and coherence by means of the full featured JavaScript programming language. For instance, to ensure that gender fields values are only limited to the strings male, female or org, we can iterate over laureate array and check if every gender value is in a hash map as in the code below:

```
var gender = Object.create({
    "male"  : true, // boolean values are unused
    "female": true,
    "org"   : true
});

for (l of laureate) {
    if (!(l.gender in gender)) {
        print("'"+l.gender+"' not allowed")
    }
}
```

While it will be MongoDB to check uniqueness of the field `_id` within the laureate collection, we are allowed to check dataset references: prize document must have valid laureates identifier:

```
// checking laureate reference
var idx = Object.create({});
for (l of laureate) {
    idx[l._id.toString()] = true;
}
for (p of prize) {
    for (l of p.laureates) {
        var id = l.id_laureate.toString();
        if (!(id in idx)) {
            print("wrong laureate id")
        }
    }
}
```

and it is also not hard to check the sharing quotes of prizes:

```
function okShare(share) {
    var num = 0; // numerator
    var den = 1; // denominator
    var i;
    for (i = 0; i < share.length; i++) {
        den *= share[i];
        var k;
        var part = 1;
        for (k = 0; k < share.length; k++) {
            if ( i != k ) {
                part *= share[k];
            }
        }
        num += part;
    }
    return num == den
}

// checking share fields
for (p of prize) {
    var vshare = [];
    for (l of p.laureates) {
        vshare.push(l.share)
    }
    if (!okShare(vshare)) {
        print("wrong share group")
    }
}
```

9.4 Populating the database

Functions similar to those presented in the previous section helps to keep documents reliable. Assuming that all of that checking code is saved in the file `check.js`, `mongo` shell is able to execute such JavaScript files with the following command:

```
$ mongo --nodb laureates.js prizes.js check.js
MongoDB shell version v4.0.1
loading file: laureates.js
loading file: prizes.js
loading file: check.js
'laureate' document to control: 916
'prize' document to control: 585
Passed 'laureate' gender fields: 916/916
Passed 'prize' laureate reference: 585/585
Passed 'prize' share fields: 585/585
```

and finally load the dataset into database with the command:

```
$ mongo laureates.js prizes.js populate.js
MongoDB shell version v4.0.1
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 4.0.1
loading file: laureates.js
loading file: prizes.js
loading file: populate.js
```

where the code of the `populate.js` JavaScript file is the following:

```
// run as
// mongo laureates.js prizes.js populate.js

// get database object
db = db.getSiblingDB("nobelprize");

// insert laureates
db.laureate.insertMany(laureate)
// insert Nobel Prize
db.prize.insertMany(prize)
```

9.5 Asking LuaTEX about laureates

In our `libmongo.lua` auxiliary library introduced in section 8 we can add a new function named `FindOne()` especially for treating the case of a query returning only one document:

```
function
lib:FindOne(collection, query, option, prefs)
    assert(type(collection) == "string")
    local client = self._client
    local coll = client:getCollection(
        self._dbname, collection
    )
    query = query or {}
    return coll:findOne(query, option, prefs)
               :value()
end
```

As a proof of concept, we can try LuaLaTEX to typeset basic information about laureates such as the birth date and place. The Lua table that corresponds to the query-generated document defines criteria with both `firstname` and `surname` fields:

```
local query = {
    firstname = "Enrico",
    surname   = "Fermi"
}
```

The complete listing below is pretty straightforward: once defined the Lua table `fermi` as a global variable, we can index it with the field key without any restriction in order to print the corresponding value everywhere in the report:

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

```
\usepackage{fontspec}
\defaultfontfeatures{Ligatures=TeX}
\setmainfont{CMU Serif}

\directlua{
local libmongo = require "libmongo"
local db = libmongo:Connect("nobelprize")
local query = {
    firstname = "Enrico",
    surname   = "Fermi"
}
fermi = db:FindOne("laureate", query)
}

\newcommand\print[1]{\directlua{
    local expr = tostring(#1);
    if expr then
        tex.print(expr)
    end
}}

\newcommand\printdate[1]{\directlua{
    local d0 = #1;
    local d1 = d0:unpack()/1000
    local d2 = os.date([[*t]], d1)
    local m = {
        "January",
        "February",
        "March",
        "April",
        "May",
        "June",
        "July",
        "August",
        "September",
        "October",
        "November",
        "December ",
    }
    tex.print(
        tostring(d2.day), m[d2.month], d2.year
    )
}}

\begin{document}
\print{fermi.firstname} \print{fermi.surname}
was born in \print{fermi.bornCity} on
\printdate{fermi.born}.
\end{document}
```

The result is:

> Enrico Fermi was born in Rome, Italy,
> on 29 September 1901.

Dates are saved as objects and not as strings. It was our initial choice. After all objects are more clever than strings because they have specific methods. When dealing with dates, we can print only the year or even the days between two dates.

In spite of that, why the source code of our working minimal example compiles under Linux but not under Windows?

It's because dates are not yet a `Date` object. In fact, the `lua-mongo` driver returns dates in BSON format. Subsequently, in the macro `\printdate` defined in the code, we call the method `unpack()` to get the date as the number of milliseconds since the Epoch[12], the moment in time fixed to the midnight of 1 January 1970.

The Lua function of the standard library `os.date()` is able to determine date values starting from the number of seconds since the Epoch, and this is the reason why in the code a factor 1/1000 multiply the unpacked value of the BSON date object.

Unfortunately, the Windows system library that deals with dates returns a null pointer if the value in time units is negative, and this is what happens in the case of Enrico Fermi, who was born before 1 January 1970. As a consequence, `os.date()` returns `nil`.

The best solution is to transform BSON date object in Lua date object. In recent time I have aimed at the LuaDate project located at `http://tieske.github.io/date/`, but I'm aware, time counting is not so easy, and this project could miss the goal depending on your need.

9.6 Asking LuaTEX about Nobel Prizes

The table 3 represents the next target. It shows the list of Nobel Prizes awarded in 2017, with Category, Laureates and Motivation of the single Prize.

Our basic model is divided into two collections: `prize` and `laureate`. In the first collection a prize document defines, along with other pieces of information, the category and, for each laureate, an identifier that refers to the corresponding document in the second collection, a share value and a motivation.

To summarise, the query process consists in three different steps:

- query prizes awarded in 2017;
- for each selected prize, query each laureate for their complete names and share values;
- then determine the main motivation as the motivation of the laureate that has the most relevant share value or, two or more laureates have the same share, that comes first in the ordered array.

MongoDB has a powerful query language with aggregate operations, projections and so on. It's probably capable to return the desired result with only one query actually composed by a chain of operations. However, I will implement the job in Lua: this paper focus isn't on learning advanced MongoDB. It is also important to notice that the ideal query takes advantage of what the server-side has to offer.

12. For more information please visit the webpage `https://en.wikipedia.org/wiki/Unix_time`.

Table 3: The Nobel Prizes awarded in 2017. The table is typeset by LuaTEX performing a direct connection to a MongoDB database as explained in the paper.

2017	
Category	Laureates (share)/Main motivation
physics	Rainer Weiss (2), Barry C. Barish (4), Kip S. Thorne (4) for decisive contributions to the LIGO detector and the observation of gravitational waves
chemistry	Jacques Dubochet (3), Joachim Frank (3), Richard Henderson (3) for developing cryo-electron microscopy for the high-resolution structure determination of biomolecules in solution
medicine	Jeffrey C. Hall (3), Michael Rosbash (3), Michael W. Young (3) for their discoveries of molecular mechanisms controlling the circadian rhythm
literature	Kazuo Ishiguro (1) who, in novels of great emotional force, has uncovered the abyss beneath our illusory sense of connection with the world
peace	International Campaign to Abolish Nuclear Weapons (ICAN) (1) for its work to draw attention to the catastrophic humanitarian consequences of any use of nuclear weapons and for its ground-breaking efforts to achieve a treaty-based prohibition of such weapons
economics	Richard H. Thaler (1) for his contributions to behavioural economics

Finally, the complete code generating the table 3 is showed in the listing below. Improvements are left to the reader, such as to eliminate the redundant `\midrule` in the last row, to capitalize the category name, or to eliminate the share value when unnecessary, that is when there is no sharing at all:

```
% !TeX program = LuaLaTeX
\documentclass{standalone}
\usepackage{fontspec}
\defaultfontfeatures{Ligatures=TeX}
\setmainfont{CMU Serif}
\usepackage{booktabs}

\newcommand\YEAR{2017}

\directlua{
local libmongo = require "libmongo"
local db = libmongo:Connect("nobelprize")
local query = {year = \YEAR}
prize = db:FindIn("prize", query)

for _, p in ipairs(prize) do
    local tl = {}
    local share, motivation
    for i, l in ipairs(p.laureates) do
        if share then
            if share < l.share then
                share = l.share
                motivation = l.motivation
            end
        else
            share = l.share
            motivation = l.motivation
        end
        local ql = {_id = l.id_laureate}
        linfo = db:FindOne("laureate", ql)
        tl[i] = linfo.firstname .. " " ..
                (linfo.surname or "") ..
                " ("..share..")"
    end
    p.laureatelist = table.concat(tl, ", ")
    p.mainmotivation = motivation or ""
end
}

\newcommand\print[1]{\directlua{
    local expr = tostring(#1);
    if expr then
        tex.print(expr)
    end
}}

\begin{document}
\begin{tabular}{lp{140mm}}
\toprule
\textbf{\YEAR}\\
\midrule
Category & Laureates (share)/Main motivation\\
\midrule
\directlua{
local bs = string.char(92)
stop = bs..bs
for _, p in ipairs(prize) do
    tex.print(p.category)
    tex.print("&")
    tex.print(p.laureatelist)
    tex.print(stop)
    tex.print("&")
    tex.print(bs.."small ")
    tex.print(p.mainmotivation)
    tex.print(stop)
    tex.print(bs.."midrule")
end
}
\bottomrule
\end{tabular}
\end{document}
```

10 Alternative ways

Some alternative ways are suitable for experimenting or for self-contained project, and can be part of a step-by-step development strategy toward an HTTP or TCP MongoDB proxy server.

Different ways to query MongoDB databases from LuaTEX are alternative in the sense that they don't use neither a Lua low-level binding nor an intermediate network service. They rely on independent tools and an elementary form of communication between LuaTEX and data, that is essentially a file.

I'm referring to a file as a communication layer in two different ways:

static: the file is saved in the file system and contains data that are a query result, encoded in a declared format like JSON;

dynamic: textual result of a query is printed to the standard output and caught by Lua via the standard function `io.popen()`.

If the channel is static, then data files are file system objects, and the unique limitation is the precise knowledge of the exchange data format. If the channel is dynamic, it is even required that tools generating data are a CLI program.

For instance, the external tool can be the `mongo` shell or Node.js. Both execute a JavaScript file or even a Rust program taking advantage of drivers like the open source project `mongo-rust-driver-prototype` written in pure Rust and providing a native interface to MongoDB.

11 Conclusion

The first part of this paper (sections 3–5) has shown how to set up a project infrastructure relying on MongoDB as database storage and LuaTEX as reporting tool. The guide is detailed both for Ubuntu and Windows, and leads you to a step-by-step source code compilation in order to build a database connector suitable for the typesetting engine.

Low-level Lua binding to the C module isn't the only communication way with MongoDB. Intermediate network services via TCP connection protocol can be implemented for LuaTEX in order to simplify the components and to ensure reliability.

The second part of this paper (sections 8–9) has shown two demo projects from the very first step in designing data models and creating database via JavaScript to the working example in LuaTEX for generating reports.

MongoDB documents are expressive, flexible and make it easy to improve data models. Thanks to the similarity to the Lua tables, high-quality reports can be typeset by LuaTEX with less code than that required when the storage engine is a SQL relational database.

Further work is required to fully use MongoDB query language capabilities, especially for building project in real context, and to make a final decision about how to safely connect LuaTEX to MongoDB.

12 Acknowledgments

I would like to thank all of the ArsTEXnica team members who have supported this work with great effort. My gratitude goes also to my family.

References

Chodorow, K. (2013). *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage*. O'Reilly Media, 2ª edizione. URL http://shop.oreilly.com/product/0636920028031.do.

Copeland, R. (2013). *MongoDB Applied Design Patterns*. O'Reilly Media, 1ª edizione. URL http://shop.oreilly.com/product/0636920027041.do.

Giacomelli, R. (2017). «A database experiment with LuajitLaTEX». *ArsTEXnica*, (23), pp. 12–34. URL http://www.guitex.org/home/numero-23.

Ierusalimschy, R. (2016). *Programming in Lua*. Lua.org, 4ª edizione.

The LuaTEX development team (2018). *LuaTEX Reference Manual*. Version 1.0.7.

▷ Roberto Giacomelli
Carrara
giaconet dot mailbox at gmail dot com

Come postare correttamente sul Forum del GUIT e vivere felici

Herr Professor Paulinho van Duck

Sommario

Esporre chiaramente il proprio problema è il primo passo per avere un aiuto rapido ed efficace, sia sul Forum del GUIT che su qualsiasi altro sito analogo. Il professor Van Duck ha trattato questo argomento sul n. 38:3 (2017) del TUGboat. Ciò che state leggendo è una versione del suo articolo riveduta e corretta per il pubblico italiano.

Abstract

Explaining in a clear way your problem is the first step to have a quick and effective help both on the GUIT Forum and on any another similar site. Professor Van Duck talked about this topic on TUGboat 38:3 (2017). What you are reading is the Italian version of his article.

1 Felice di conoscervi!

Prof. van Duck

Ciao a tutti!
Sono Herr Professor Paulinho van Duck, sono nato a San Paolo del Brasile ma ora vivo a Milano, dove condivido un appartamento con la mia amica Carla. Lei mi aiuta anche a scrivere i miei documenti e a rispondere alle email, poiché non è facile usare una tastiera quando non si dispone di un becco appuntito!

Quelli di voi che che leggono il TUGboat o che frequentano il sito di Domande & Risposte TEX StackExchange (TEX.SE) probabilmente mi conoscono già.[1] Anche i partecipanti all'ultimo GUIT*meeting* forse si ricordano del piccolo anatroccolo presente accanto al mitico prof. Enrico Gregorio durante il suo intervento. Beh, quello ero io, *quack!*

Io non sono un grande esperto di LATEX, però ne sono entusiasta e sono contento di poter aiutare i principianti come me. Claudio Beccari, uno dei pilastri del GUIT, ha molto apprezzato il mio primo articolo apparso sul TUGboat (lo ringrazio per questo) e mi ha chiesto di farne una versione italiana.

Figura 1: Esempio di domanda mal posta. Del tipo *fatelo-per-me*.

L'articolo spiega come impostare correttamente una domanda per il sito TEX.SE, in particolare come creare un MWE (*Minimal Working Example*), ovvero un EMC (Esempio Minimo Compilabile): un documento, completo e il più sintetico possibile, che riproduca il problema.

Devo dire che gli utenti italiani sono facilitati in quanto il Forum del GUIT non ha le rigide regole di TEX.SE, che a volte risultano spiazzanti per i novellini.[2] Nonostante ciò, la creazione di un esempio minimo che riproduca il problema che state incontrando è senza dubbio utile per ottenere velocemente una soluzione.

Vediamo qualche trucchetto per facilitarvi il compito.

2 Cosa *non* postare!

Iniziamo mostrando il modo sbagliato di porre una domanda. Nella figura 1 possiamo vedere un esempio con numerosi difetti: titolo/oggetto generico, seguito da una figura (o tabella, o chi più ne ha più ne metta) più o meno complicata, senza una minima spiegazione del problema né uno straccio di riga di codice.

Queste domande sono chiamate *fatelo-per-me* (*just-do-it-for-me*, nel gergo di TEX.SE), perché non mostrano nessuno sforzo da parte di chi le pone. Per favore, evitatele, un piccolo anatroccolo piange quando vede queste cose!

1. `https://tex.stackexchange.com/`

2. `http://www.guitex.org/forum`.

Perché non funziona?

Il mio codice fino a ieri funzionava, invece adesso mi dà un errore. Perché?

Figura 2: Esempio di domanda mal posta. Del tipo *sfera di cristallo*.

Un altro esempio simile sono le domande *sfera-di-cristallo*, come quella della figura 2. Sono chiamate così perché solo un indovino riuscirebbe a rispondere. Dato che non conosco TEXnici che sappiano leggere nel pensiero, cercate sempre di essere chiari nell'esporre il vostro problema e di fornire tutti i dati necessari per identificarlo.

Al contrario, gli utenti 'svegli' seguono sempre queste linee guida:

Le Leggi di van Duck

1. Leggere i manuali dei pacchetti.
2. Guardare il log.
3. Cercare su Internet.
4. Allegare sempre un Esempio Minimo Compilabile alla domanda.

L'ultima regola merita di essere esaminata nel dettaglio. Solo poche domande non richiedono un Esempio Minimo Compilabile. Per esempio, se state chiedendo qualcosa riguardo alle impostazioni di un editor, probabilmente un EMC non serve, ma in genere è *indispensabile*. Purtroppo succede spesso che non venga allegato alla domanda, specialmente se chi la posta è un principiante.

Il mio studio accurato del fenomeno mi ha portato a enunciare la famosa Equazione di van Duck:

L'Equazione di van Duck

$$U_b = U_k + U_h + U_l$$

dove gli utenti che pongono una domanda mal formulata (U_b, il pedice b sta per *bad* question), senza un EMC, sono suddivisi in tre categorie:

U_k = quelli che non hanno idea di cosa sia un EMC (they don't *know*)
U_h = quelli che ne hanno sentito parlare ma non sanno come costruirlo (they don't know *how*)
U_l = quelli che sanno benissimo cos'è ma sono troppo pigri per aggiungerlo (they are *lazy*).

Escludendo i pigroni, che sono una battaglia persa in partenza, per tutti gli altri spero che queste mie righe siano utili.

3 Cosa dovreste fare *prima* di postare una domanda

Uno dei vantaggi di TEX & Co. è l'abbondante documentazione, approfittatene!

So che leggere i manuali dei pacchetti è noioso, a volte addirittura impossibile. Presumo che nessuno al mondo abbia letto le più di 1000 pagine del manuale di Ti*k*Z & PGF, nemmeno l'autore dopo che l'ha scritto, *quack!* Comunque, le informazioni riguardanti le incompatibilità o le precauzioni da prendere per caricare il pacchetto prima o dopo altri, di solito, sono scritte all'inizio della documentazione. Si può anche cercare all'interno del pdf della documentazione quella particolare opzione che ci serve e leggere solo la sua spiegazione o scorrere l'indice analitico per trovare il comando di cui abbiamo scordato il nome. Insomma, una rapida occhiata alla documentazione è d'obbligo!

Nella Parte I del manuale di Ti*k*Z, per esempio, c'è un ottimo tutorial. È sufficiente leggerlo per cominciare ad adoperare questo pacchetto (terrorizzante per i neofiti).

Un altro strumento fondamentale è il log, soprattutto in caso di errore. OK, le descrizioni degli errori di TEX non sono il massimo in fatto di chiarezza. Un principiante di solito rimane un po' perplesso scoprendo che `Undefined control sequence` significa semplicemente che c'è un errore di battitura nel nome di un comando o che il relativo pacchetto non è stato caricato.

Comunque, se cercate il vostro misterioso messaggio di errore su Internet, vedrete che quasi sempre troverete la soluzione o, perlomeno, ne capirete il significato.

Nel log è indicata anche la posizione in cui l'errore è stato incontrato, (quasi sempre) corrisponde con la linea di codice dove è presente la *control sequence* errata.

Infine, ricordatevi che l'errore più importante è il primo, gli altri potrebbero essere una conseguenza.

Non vi insegnerò come debuggare il vostro codice, in questa occasione. Per chi fosse interessato, c'è un fantastico articolo di Barbara Beeton sull'argomento (TUGboat 38:2, p. 159, `tug.org/TUGboat/tb38-2/tb119beet.pdf`) tradotto in italiano su ArsTEXnica 25. Invece, mi piacerebbe farvi scoprire un comando molto comodo: `\listfiles`. Aggiungendolo al vostro documento, troverete nel log la lista e le versioni di tutti pacchetti che state utilizzando. Ciò è molto utile in casi come: *Perché*

questo codice funziona sul computer del mio amico ma non sul mio? oppure *Perché funziona su ShareLaTeX ma non su Overleaf?*[3] Probabilmente perché ci sono differenti versioni dei pacchetti! Molti problemi possono essere risolti semplicemente aggiornando la vostra distribuzione di TeX (ovvero TeX Live, MiKTeX o simili).

Giusto per vedere come funziona, compilate questo semplice codice sorgente:

```
\listfiles
\documentclass{article}
\usepackage{tikzducks}
\begin{document}
  \begin{tikzpicture}
    \duck
  \end{tikzpicture}
\end{document}
```

nel vostro log, date un'occhiata alle righe comprese tra:

```
*File List*
...
***********
```

troverete molti pacchetti elencati, con la relativa versione indicata a fianco. Notate che, nell'esempio, viene caricato solo tikzducks ma il log elenca tutti i pacchetti richiamati da tikzducks stesso![4]

A volte è fondamentale saperlo, perché vi permette di evitare di caricare pacchetti due volte, a meno che non abbiate la necessità di impostare una particolare opzione, e perché facilita l'identificazione di una incompatibilità tra pacchetti "nascosta".

Un'altra preziosa fonte di informazioni è Internet. Se avete un problema, è possibile che qualcun altro lo abbia avuto prima di voi: fate una ricerca col vostro browser, è molto probabile che una risposta appaia. Questo consiglio può sembrare banale, ma vi assicuro che molte volte vedo domande già presenti in tutte le TeX FAQ del mondo a cui chiunque potrebbe rispondere semplicemente cercando in rete, *quack!* Comunque, state attenti: come qualsiasi altra notizia che trovate su Internet, qualche informazione potrebbe essere non del tutto corretta o essere obsoleta. Il classico esempio è il comando `\rm`, che è sconsigliato da più di vent'anni (in matematica si deve usare `\mathrm`, nel testo `\textrm` al suo posto) ma appare spesso qua e là in rete.

Prima di decidere di utilizzare qualsiasi pacchetto o comando, consultate almeno un paio di fonti. Nella pagina dedicata alla documentazione del sito del GuIt[5] potete trovare un elenco di risorse affidabili.

4 Il modo *corretto* di chiedere

Se tutti gli sforzi per risolvere il vostro problema in autonomia sono stati infruttuosi, è arrivato il momento di chiedere sul Forum.

Per prima cosa controllate che non esista già un thread simile, magari con la risposta già pronta!

Poi scegliete quale sezione del Forum è più adatta al vostro problema, ne esistono diverse:

- TeX Help, che comprende: il forum sui problemi riguardanti LaTeX e TeX, quello per ConTeXt e quello sugli altri programmi che si utilizzano congiuntamente a TeX/LaTeX (tipicamente, gli editor dedicati).
- Discussioni avanzate: problemi di tipografia in generale, gestione della didattica e segnalazione di corsi, edizioni critiche.
- Gestione del GuIt, dove sono annunciati anche i meeting.

Cercate di evitare l'off topic, ovvero messaggi che hanno poco o niente a che vedere con gli argomenti trattati nel Forum.

Vediamo ora come impostare la vostra domanda.

Prima di tutto l'oggetto: non siate vaghi. Non scrivete *Come si fa questo in LaTeX?* oppure *Perché il mio codice mi dà un errore?* Tenete conto che le persone che rispondono sono volontari che lo fanno nel loro (limitato e prezioso) tempo libero, costringerli ad aprire il messaggio per vedere di che si tratta è una perdita di tempo. Qualche utente, per di più, legge solo i messaggi a cui pensa di poter rispondere (guardando l'oggetto). Se il titolo è generico, potreste perdere l'occasione di avere una risposta rapida. Inoltre, non siate egoisti, *quack*! Pensate a chi dopo di voi avrà lo stesso problema, dategli la possibilità di trovare il vostro messaggio facilmente.

In secondo luogo, nel corpo del vostro messaggio indicate tutte le informazioni necessarie per comprendere il vostro problema: eventuale messaggio d'errore, sistema operativo e codifica utilizzati, tipo di compilazione e così via. Non allegate solo il codice o, ancora peggio, solo un'immagine.

3. Per gli interessati, ShareLaTeX (`www.sharelatex.com`) e Overleaf (`v2.overleaf.com`) sono due siti che consentono di usare LaTeX online, senza la necessità di installare una distribuzione sul proprio computer. Le due società, tra l'altro, si stanno fondendo, ed è già disponibile una versione sperimentale del futuro sito comune: `v2.overleaf.com`

4. `https://ctan.org/pkg/tikzducks`

5. `https://www.guitex.org/home/it/doc`

Infine, arriviamo al punto più importante: allegare un Esempio Minimo Compilabile, se il problema lo richiede (in genere il 99,9% dei casi). Vediamo rapidamente i passi essenziali da seguire per crearlo (per maggiori dettagli potete consultare la guida di Beccari[6]).

Prima del codice vero e proprio, sarebbe comodo indicare la codifica utilizzata (UTF-8, Latin-1, ecc.) e con quale programma avete eseguito la compilazione (pdflatex, lualatex o xelatex), attraverso le *righe magiche*:

```
\% !TEX encoding = ...
\% !TEX TS-program = ...
```

Poi non dimenticate di partire con la `\documentclass` che state utilizzando. È importante indicarla, anche se state chiedendo qualcosa riguardante una tikzpicture o un'espressione matematica. La risposta potrebbe cambiare, ad esempio, a seconda che stiate usando beamer o article.

Poi i pacchetti: inserite tutti i pacchetti necessari per riprodurre il vostro problema, e solo quelli, non siate prolissi! E non dimenticate le opzioni che avete impostato.

Riportate le eventuali macro che avete creato con `\newcommand`, così come gli eventuali nuovi ambienti.

Lo stesso ragionamento vale per il codice. Ponetelo tra:

```
\begin{document}
...
\end{document}
```

e aggiungete tutto e solo quanto strettamente necessario per riprodurre il vostro problema. Non allegate solo pezzetti di codice.

Ricordate di testare il vostro EMC prima di allegarlo alla domanda. Dovete essere sicuri che si possa compilare riproducendo il vostro problema o, se c'è un errore di compilazione, che sia lo stesso su cui vi state scervellando.

Vi assicuro che creare un Esempio Minimo non è solo utile per chi vorrà aiutarvi ma, credetemi, anche per voi. Non so quante volte, mentre stavo preparando un EMC, ho trovato la soluzione da solo, *quack*!

Esistono anche diversi pacchetti che vi aiutano nella creazione di un EMC.

6. Disponibile all'indirizzo `http://www.guitex.org/home/images/doc/GuideGuIT/guidaemc.pdf`

Alcuni pacchetti come lipsum[7] e blindtext[8] vi consentono, nel caso fosse utile per mostrare l'errore, di produrre delle righe di testo senza significato, senza dover inserire il testo reale, quindi evitando problemi di privacy o di copyright, e senza doversi inventare delle parole a caso.

Il pacchetto graphicx[9] vi permette di usare delle immagini di esempio al posto delle originali.

Esiste persino mwe[10] (il nome è già tutto un programma) che raccoglie i pacchetti più comunemente utilizzati per creare degli esempi minimi e fornisce una raccolta di immagini di prova.

Se poi volete essere spiritosi, provate duckuments,[11] l'analogo di mwe per i fan delle papere (intese come anatre, non come vistosi errori involontari).

Un altro pacchetto molto comodo, da utilizzare in fase di test, è showframe[12] che evidenzia i contorni delle varie parti della vostra pagina: testatina, corpo, margini, eccetera. È utile per individuare le famigerate `Overfull \hbox` e, in generale, per perfezionare l'allineamento. È sufficiente inserirlo nel preambolo e compilare. Per esempio, il seguente codice genera un `Overfull \hbox`:

```
\documentclass{book}
\usepackage{showframe}
\usepackage{mwe}
\begin{document}
 \blindtext

 \includegraphics[width=\linewidth]{%
    example-image-a}

 \blindtext
\end{document}
```

Se lo compilate, otterrete l'output mostrato nella figura 3. Come potete notare, l'errore di rientro è chiaramente visibile e rapidamente risolvibile mettendo un `\noindent` prima di `\includegraphics`.

Se il problema che state riscontrando riguarda i riferimenti bibliografici, dovreste anche allegare alla domanda il vostro file `.bib`. Ovviamente, non dovete inserire il file completo, ma solo i `riferimenti bibliografici` necessari a riprodurre l'errore.

La cosa migliore sarebbe creare un Esempio Minimo Compilabile con Bibliografia (EMCB), con il file `.bib` incluso direttamente nel codice (in questo

7. `https://ctan.org/pkg/lipsum`
8. `https://ctan.org/pkg/blindtext`
9. `https://ctan.org/pkg/graphicx`
10. `https://ctan.org/pkg/mwe`
11. `https://ctan.org/pkg/duckuments`
12. `https://ctan.org/pkg/showframe`

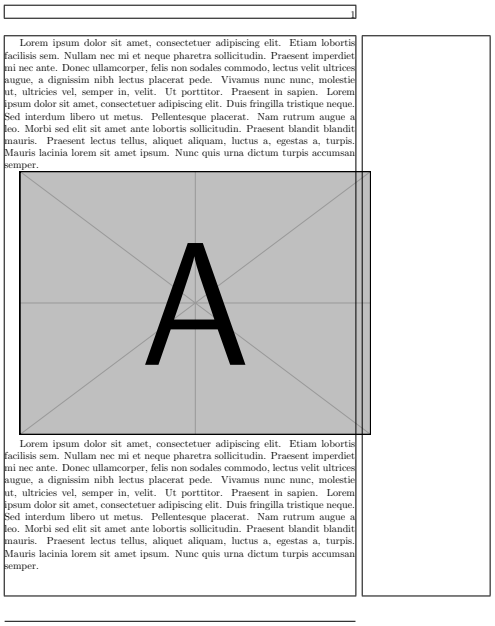


Figura 3: Esempio d'uso dei pacchetti mwe e showframe. La causa dell'`Overfull \hbox` è immediatamente identificabile (rientro).

modo, chi volesse aiutarvi, dovrebbe solo copiare e incollare il vostro esempio e compilarlo; ricordatevi il discorso sul volontariato che ho fatto prima).

L'ambiente filecontents*[13] è quello che fa per voi. È anche preferibile, invece di inventare un nome per il vostro file `.bib` di test, usare `\jobname.bib`: in questo modo gli viene automaticamente assegnato lo stesso nome del vostro file `.tex`, cambiando solo il suffisso. Qui trovate un semplice schema da seguire:

```
\begin{filecontents*}{\jobname.bib}
% inserite qui i vostri riferimenti
% bibliografici strettamente necessari
...
\end{filecontents*}
\documentclass{...}
% caricate qui i vostri pacchetti
% compreso quello per gestire la
% bibliografia
...
% con biblatex:
\addbibresource{\jobname.bib}

\begin{document}
% inserite qui il vostro testo con le
% citazioni
...
% e per stampare la bibliografia, con
% biblatex:
\printbibliography
% o con un altro pacchetto
% bibliografico:
\bibliography{\jobname}
\end{document}
```

In alternativa, se il vostro problema non riguarda in particolare i vostri `bibitem`, potete utilizzare un file di test che è incluso nel potente pacchetto biblatex:[14] `biblatex-examples.bib`.

Il trucchetto dell'ambiente filecontents* (o equivalente) dovrebbe essere utilizzato per qualsiasi file di testo che serve all'esempio minimo, da un pacchetto personalizzato (file con estensione `.sty`) a un `.txt` o un `.dat` da leggere con csvsimple, pgfplots/pgfplotstable o simili.

Infine, dopo aver postato la domanda, non abbandonate il Forum, ricordatevi di aver aperto un thread e rispondete alle eventuali richieste di chiarimento.

Quando avrete la risposta che fa per voi, aggiungete al thread un riassunto che la contenga.

5 Conclusioni

Spero che i miei consigli vi siano stati utili. Se li seguirete, sono certo che la soluzione al vostro problema non tarderà ad arrivare, *quack!*

▷ Herr Professor Paulinho van Duck
Duck University Campus
Stagno delle Anatre
Parco Sempione
Milano
paulinho.vanduck@gmail.com

13. Esiste anche un pacchetto filecontents, `https://ctan.org/pkg/filecontents`

14. `https://ctan.org/pkg/biblatex`

Aggiornamento di arara

Claudio Beccari, Paulo Roberto Massa Cereda

Sommario

Nel numero 25 di ArsTEXnica veniva annunciata la nuova versione di arara. Il programma era già disponibile allora, ma era privo della documentazione. Ora la distribuzione aggiornata e completa di TEX Live contiene sia il software sia la nuova documentazione.

Abstract

On issue 25 of ArsTEXnica the updated version of arara was announced as imminent. Its software was already available, but its documentation was missing. Now the updated complete distribution of TEX Live contains the new software and the new documentation.

Questo breve articolo serve sostanzialmente per aggiornare quello precedente apparso su ArsTEXnica 25, Massa Cereda e Beccari (2018), dove venivano brevemente richiamati il programma e il linguaggio di arara per mostrare come usarli per un uso insolito. Venivano mostrate alcune 'regole' scritte nel precedente linguaggio di arara 3.x, ma gli autori si ripromettevano di inviare un aggiornamento non appena arara 4.x fosse stato disponibile.

Questo è l'aggiornamento promesso.

In realtà, le premesse del nuovo linguaggio vanno studiate sulla sua documentazione, perché esso vorrebbe essere più *user friendly* di quello usato in arara 3.x; gli autori di arara 4.0, che ora sono un team, lo sperano veramente, ma solo leggendo la documentazione in Cereda *et al.* (2018) ci si può rendere conto dei passi avanti che il software ha compiuto.

Qui ci limitiamo a fornire al lettore i nuovi file per gestire l'uso insolito trattato in (Massa Cereda e Beccari, 2018).

Il codice seguente sostituisce quello riportato nel Codice 1 di (Massa Cereda e Beccari, 2018).

```
!config
identifier: writeconfig
name: WriteConfig
authors:
- Claudio Beccari
- Paulo Cereda
commands:
- name: Setting the paper size
  command: >
    @{
        dict = [ 'A4' : 'aquattro',
                 'A5' : 'acinque',
                 'B5' : 'bcinque' ];

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

        writeToFile(getBasename(file) + '.cfg', text, false);
        writeToFile('suffix.cfg', suffix, false);

        return true;
    }
arguments:
- identifier: suffix
  flag: >
    @{
        papers = [ 'A4', 'A5', 'B5' ];
        i = 1;
        if (parameters.suffix == 'ask') {
            i = showDropdown(4, 'Paper size', 'Please select the paper size.',
                papers.toArray());
```

```
            if (i == 0) {
                throwError('You have to select the paper size!');
            }
            return papers[ i - 1];
        }
        else {
            if (!papers.contains(parameters.suffix)) {
                throwError('You selected an unsupported paper size!');
            }
            return parameters.suffix;
        }
    }
  default: 'A4'

```

Invece questo codice sostituisce quello indicato nel Codice 2 di (Massa Cereda e Beccari, 2018).

```
!config
identifier: rename
name: Rename
authors:
- Paulo Cereda
- Claudio Beccari
commands:
- name: The rename operation
  command: >
    @{
        suffix = readFromFile('suffix.cfg')[0];
        base = getBasename(file);
        return getCommand('cp', base + '.pdf', base + suffix + '.pdf');
    }
arguments: []
```

Se si confrontano i codici vecchi e nuovi non si incontrano differenze sostanziali, ma quelli nuovi sono un po' più semplici; le righe sono un poco più numerose perché essi hanno prestazioni più ampie. Un piccolo esempio fra gli altri: la prima regola, se non è invocata in modo corretto, si ferma e chiede all'utente di inserire subito la sigla del formato di carta dimenticata.

Riferimenti bibliografici

Cereda, P., Daniel, M., Longborough, B. e Talbot, N. (2018). «arara– The cool TEX automation tool». La documetazione è leggibile con `texdoc arara`.

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

Massa Cereda, P. R. e Beccari, C. (2018). «Un uso insolito di arara ». *ArsTEXnica*, (25). URL https:www.guitex.org/home/images/ArsTeXnica/AT025/arstexnica25.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`

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

Versione elettronica per la diffusione via web.

ArsTEXnica – Call for Paper

La rivista è aperta al contributo di tutti coloro che vogliano partecipare con un proprio articolo. Questo dovrà essere inviato alla redazione di ArsTEXnica, per essere sottoposto alla valutazione di recensori entro e non oltre il 14 Febbraio 2019. È necessario che gli autori utilizzino la classe di documento ufficiale della rivista; l'autore troverà raccomandazioni e istruzioni più dettagliate all'interno del file d'esempio (.tex).

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

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

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

ArsTEXnica

Rivista italiana di TEX e LATEX

Numero 26, Ottobre 2018