

Numero 14
Ottobre
2012

ArsT_EXnica

Rivista italiana di T_EX e L^AT_EX

Atti del G_UIT*meeting* 2012

G_UIT

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



G_UI_T – Gruppo Utilizzatori Italiani di T_EX

ArsT_EXnica è la pubblicazione ufficiale del G_UI_T

Comitato di Redazione

Gianluca Pignalberi – *Direttore*

Renato Battistin, Claudio Beccari

Marco Buccino, Riccardo Campana,

Massimo Caschili, Gustavo Cevolani

Massimiliano Dominici, Andrea Fedeli

Tommaso Gordini, Enrico Gregorio

Carlo Marmo, Lapo Mori

Antonello Pilu, Ottavio Rizzo

Gianpaolo Ruocco, Emmanuele Somma

Enrico Spinielli, Emiliano Vavassori

Emanuele Vicentini, Raffaele Vitolo

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

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

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

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

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

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

arstexnica@sssup.it.

Nota sul Copyright

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

① **Attribuzione:** devi riconoscere il contributo dell'autore originario.

© **Non commerciale:** non puoi usare quest'opera per scopi commerciali.

© **Non opere derivate:** non puoi alterare, trasformare o sviluppare quest'opera.

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

Per maggiori informazioni:

<http://www.creativecommons.org>

Associarsi a G_UI_T

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

Indirizzi

Gruppo Utilizzatori Italiani di T_EX:

c/o Ufficio Statistica

Scuola Superiore Sant'Anna

Piazza Martiri della Libertà 33

56127 Pisa, Italia.

<http://www.guitex.org>

guit@sssup.it

Redazione ArsT_EXnica:

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

arstexnica@sssup.it

Codice ISSN 1828-2369

Stampata in Italia

Pisa: 15 Ottobre 2012

GUI meeting 2012

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

27 ottobre 2012

Napoli, Centro Congressi Partenope, via Partenope, 36

Aula Magna

Università degli Studi di Napoli "Federico II"

Programma del convegno

- 09:00 Benvenuto. Inizio dei lavori.
- 09:10 *Beyond BibT_EX*. Jean-Michel Huppen.
- 09:40 *La forma del testo umanistico: la classe sufsesi*. Ivan Valbusa.
- 10:10 *Una classe per comporre un dizionario*. Claudio Beccari, Heinrich Fleck.
- 10:40 — Pausa caffè (30').
- 11:10 *L^AT_EX3: un nuovo gioco per i maghi e per diventarlo*. Enrico Gregorio.
- 11:40 *Un pacchetto per il controllo del codice fiscale italiano*. Claudio Beccari.
- 12:10 *Grafica ad oggetti con LuaT_EX*. Roberto Giacomelli.
- 12:40 *MFLua: Instrumentation of METAFONT with Lua*. Luigi Scarso.
- 13:10 — Pausa pranzo (80').
- 14:30 *Gestione 'quasi automatica' dei valori numerici in esempi di calcolo ed esercizi svolti*. Agostino De Marco.
- 15:00 *Il pacchetto esami per la creazione di prove scritte*. Grazia Messineo, Salvatore Vassallo.
- 15:30 *Test on line di matematica: il Progetto Minerva*. Grazia Messineo, Salvatore Vassallo.
- 16:00 — Pausa caffè (30').
- 16:30 *Stemma codicum con L^AT_EX*. Matteo Fadini.
- 17:00 *Composizione di uno Stemma codicum con TikZ*. Jerónimo Leal, Gianluca Pignalberi.
- 17:30 *Alcuni stralci sul Computer Bodoni*. Claudio Vincoletto.
- 18:00 Chiusura dei lavori. Arrivederci.

La partecipazione all'evento è libera e gratuita.

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

Informazioni: prof. Agostino De Marco, agostino.demarco@unina.it



www.guitex.org



UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II



Dipartimento di Ingegneria Aerospaziale

ArsT_EXnica

Rivista italiana di T_EX e L^AT_EX

Numero 14, Ottobre 2012

Editoriale	
<i>Gianluca Pignalberi</i>	5
Beyond BIBT _E X	
<i>Jean-Michel Hufflen</i>	7
La forma del testo umanistico: la classe <code>suftesi</code>	
<i>Ivan Valbusa</i>	15
Una classe per comporre un dizionario	
<i>Claudio Beccari, Heinrich Fleck</i>	31
L ^A T _E X3: un nuovo gioco per i maghi e per diventarlo	
<i>Enrico Gregorio</i>	41
Un pacchetto per il controllo del codice fiscale italiano	
<i>Claudio Beccari</i>	48
Grafica ad oggetti con LuaT _E X	
<i>Roberto Giacomelli</i>	53
MF _L ua: Instrumentation of METAFONT with Lua	
<i>Luigi Scarso</i>	72
Gestione ‘quasi automatica’ dei valori numerici in esempi di calcolo ed esercizi svolti	
<i>Agostino De Marco</i>	82
Il pacchetto <code>esami</code> per la creazione di prove scritte	
<i>Grazia Messineo, Salvatore Vassallo</i>	95
Test online di matematica: il Progetto M.In.E.R.Va	
<i>Grazia Messineo, Salvatore Vassallo</i>	104
<i>Stemma codicum</i> con L ^A T _E X	
<i>Matteo Fadini</i>	113
Composizione di uno <i>Stemma codicum</i> con TikZ	
<i>Jerónimo Leal, Gianluca Pignalberi</i>	123
Alcuni stralci sul Computer Bodoni	
<i>Claudio Vincoletto</i>	131

Gruppo Utilizzatori Italiani di T_EX

Editoriale

Gianluca Pignalberi

Eppur si muove! mi verrebbe da dire. Il `Gftrmeeting` è a Napoli, per la prima volta fuori e lontano da Pisa. Già questo è un piccolo, bell'avvenimento indice del fatto che la comunità `TeX` italiana è dislocata ovunque, e ovunque ci tiene a riunirsi. L'altro piccolo, bell'avvenimento/movimento è dato dal numero di interventi: tredici articoli presentati e due in incubazione è un risultato senza precedenti, specie se verificato dopo un meeting più che in sordina, quello col più basso numero di interventi, sebbene di assoluto rilievo, e rimpolpato con alcune presentazioni tratte da un `ArsTeXnica` ordinario.

Tutto si muove, come potete vedere dal nuovo sito e dalla quantità di materiale che viene costantemente aggiunto. Anche il manifesto del meeting si muove e abbandona il leoncino (che per fortuna non è reale e quindi non si aggirerà per le nostre città mendicando una carezza, dell'acqua e un po' di cibo come parecchi dei cani che di nuovo hanno abbandonato nella mia città e che nessuno adotterà). Sono tutti piccoli segni di un continuo movimento (se fossimo un fumetto sarebbero delle linee cinetiche) e rinnovamento. Cos'altro non ho menzionato? Lo sviluppo del kit degli autori di `ArsTeXnica` passa su GITHUB e diventa collaborativo, alcune persone esterne al `Gftr` offrono spontaneamente il loro lavoro, la pagina del lavoro è una realtà, sebbene ancora ignorata da chi dovrebbe reclutare esperti di `TeX`. Anche `LATeX` stesso è in movimento, cosa di cui parleremo a breve.

Uh, dimenticavo... i nuovi iscritti non avranno più la spilletta col leoncino. Il regalo di benvenuto viene sostituito con un tipometro nuovo fiammante. Il disegno, a cura di Massimo Caschili, è lo stesso del vecchio tipometro, ma il materiale su cui è stampato è di qualità infinitamente superiore, un vero e proprio invito a usarlo il più spesso possibile.

Passiamo ai contributi del meeting e di questo numero di `ArsTeXnica` che ne raccoglie gli atti.

Jean-Michel Huffle pone ancora una volta la sua attenzione e la sua scienza al servizio delle bibliografie. Questa volta è il turno di `mlbiblatex` e `mlbibcontext`, il cui uso fa superare le limitazioni imposte da `BibTeX` per i due noti formati `LATeX` e `ConTeXt`.

Ivan Valbusa presenta la sua classe per documenti dichiaratamente umanistici: `suftesi`. Guardate le immagini, anche senza leggere l'articolo, e apprezzatene la pregevolezza. I *referee* si sono sbilanciati a preconizzare la "fine" di `ClassicThesis`. Avranno sbagliato previsione? Non credo.

Claudio Beccari, insieme a Heinrich Fleck, ci propone una soluzione al problema della composizione di un dizionario. I due autori mostrano dei metodi interessanti per la determinazione dell'unghia (la lettera guida stampata in grande a bordo pagina necessaria a localizzare velocemente un intervallo di ricerca), per il posizionamento delle tabelle oltre il margine e il posizionamento dei medaglioni, per il trattamento dei lemmi e per la determinazione dei *mark*.

Enrico Gregorio delizia i lettori con il suo ottimo articolo su `LATeX3`, formato che promette di soppiantare in un attimo (appena sarà terminato) la potenza espressiva di `LATeX 2ε`. L'articolo mostra alcune funzioni del nucleo di `LATeX3`, con particolare attenzione alle variabili di tipo *sequence* e al modulo `l3keys`.

Ancora Claudio Beccari presenta il suo pacchetto per il controllo del codice fiscale, pacchetto in cui affronta e risolve tutte le operazioni necessarie a verificare che il codice fiscale immesso risponda alle regole stabilite. Tutti i calcoli sono demandati a `TeX`, nessuno escluso. L'importanza dell'articolo, come ammette l'autore, sta nel mostrare come `TeX` può risolvere quei calcoli presenti in molti ambiti lavorativi (e non).

Roberto Giacomelli ci mostra la generazione di oggetti grafici sfruttando gli oggetti di `LuaTeX`. Siccome `LuaTeX` ingloba all'interno di `TeX` l'interprete Lua, è possibile scrivere del codice Lua in un sorgente `TeX`. Lua è un linguaggio di programmazione orientato agli oggetti, un paradigma diverso rispetto a quello di `TeX`. Giacomelli esamina e spiega come questa commistione modifichi l'approccio al disegno grafico.

Luigi Scarso ci offre un'esperienza reale fatta con il suo `MFLua`. L'articolo, che cela una caterva di informazioni geometriche e di considerazioni grafiche, mostra come `MFLua` aiuta a ottenere dei font vettoriali a partire da un codice `METAFONT`.

Agostino De Marco disserta sulle tecniche per automatizzare la generazione di esercizi contestualizzati a partire da un esempio generico. Dato un tipo di esercizio vengono generati più esempi concreti variando i dati numerici, realizzandone i calcoli e impaginandone i risultati, tutto automaticamente.

La coppia Grazia Messineo/Salvatore Vassallo condivide la propria esperienza di preparazione di documenti per gli esami in due interessantissimi articoli dedicati alla generazione dei testi d'esame, uno "su carta" e l'altro online. Presentano un pacchetto da essi realizzato proprio per risol-

vere queste situazioni e per aumentare la potenza espressiva dei pacchetti preesistenti.

Matteo Fadini concentra la sua attenzione sulla costruzione di uno *stemma codicum* prendendo in esame i pacchetti *synttree* e *xytree*. Mostra alcuni esempi anche molto complessi. Jerónimo Leal e Gianluca Pignalberi rispondono a quest’articolo mostrando che *TikZ* permette di ottenere risultati visivamente più gradevoli senza per questo imporre una notazione più difficile, anzi uniformandola sia nel caso in cui vengano descritti degli alberi che nel caso in cui bisogna descrivere dei grafi.

Infine Claudio Vincoletto prosegue il suo studio sui font dopo la sfortunata esperienza del *Mar-*

dersteig, quel font che doveva essere la versione libera di Dante, mai autorizzata da Adobe e di cui Martino Mardersteig detiene in esclusiva la versione “valdonegana”. In questo articolo Vincoletto mostra le tecniche usate per adattare il sorgente METAFONT di Computer Modern e renderlo un font dalle caratteristiche “bodoniane”: Computer Bodoni.

Questo è tutto. Buon meeting e buona lettura.

▷ Gianluca Pignalberi
g dot pignalberi at
alice dot it

Beyond Bib_TE_X

Jean-Michel Huffer

Abstract

This article is related to the production of bibliographies for documents typeset by means of L^AT_EX or ConT_EXt. We explore some recent ways to go beyond what is allowed by Bib_TE_X. Then we explain why the programs `mlbiblatex` and `mlbibcontext` seem to us to be promising.

Keywords Bib_TE_X, MIBib_TE_X, `mlbiblatex`, `mlbibcontext`, L^AT_EX, ConT_EXt MkII, MkIV, Lua_TE_X, `biblatex` package, `bib` module.

Sommario

Questo articolo tratta la produzione di bibliografie di documenti composti con L^AT_EX e ConT_EXt. Esploriamo alcune tecniche recenti che soppiantano quanto permesso da Bib_TE_X. Quindi illustriamo perché i programmi `mlbiblatex` e `mlbibcontext` ci sembrano promettenti.

Parole chiave Bib_TE_X, MIBib_TE_X, `mlbiblatex`, `mlbibcontext`, L^AT_EX, ConT_EXt MkII, MkIV, Lua_TE_X, pacchetto `biblatex`, modulo `bib`.

Introduction

For many years, Bib_TE_X (Patashnik, 1988a) was unrivalled as the bibliography processor associated with the L^AT_EX typesetting system. Let us recall that Bib_TE_X extracts *bibliographical keys* from an auxiliary (.aux) file built by L^AT_EX: let *f* be a file name without suffix, the command ‘`bibtex f`’ is equivalent to ‘`bibtex f.aux`’. This .aux file also provides a list of *bibliography database* (.bib) files to be searched. The corresponding *entries* are extracted, formatted according to a *bibliography style*, and the result is a *f.bbl* file of *bibliographical references*¹ for the *f.tex* document. Most of bibliography styles sort the relevant references according to the conventions they are designed to enforce. The ones that do not are *unsorted*, they just list these references according to the order of first citation throughout the document.

However, Bib_TE_X is ageing: the current version has been in use for more than a decade. More precisely, its bibliography styles (.bst files) are writ-

ten using an old-fashioned language using post-fixed notations and based on manipulating a stack (Patashnik, 1988b); new requirements have appeared during the last decade (Huffer, 2011)—multilingualism, encodings,² other typesetting systems built out of T_EX, e.g., ConT_EXt (Hagen, 2001)—beyond the capabilities of this venerable program.

Another approach has been recently proposed whereby references are stored into .bbl files as *structures*, and formatting ‘References’ sections is entirely deferred to the typesetting system. The L^AT_EX commands of the `biblatex` package³ (Lehman, 2011) are in charge of such formatting. A similar framework had been put into action by Taco Hoekwater’s `bib` module of ConT_EXt (Hoekwater, 2001). Even if the `biblatex` package’s functions may be used with Bib_TE_X as bibliography processor—in which case the `biblatex` bibliography style is applied—another bibliography processor, more recent, is recommended: `biber` (Charette and Kime, 2011). This bibliography processor is a good replacement for Bib_TE_X, although it is quite slow—it has been developed using Perl⁴—but can generate only references suitable for the `biblatex` package, not for ConT_EXt’s `bib` module.

In the next section, we explain why the difficulty of writing a new bibliography processor from scratch is mainly related to the format of bibliography database (.bib) files. Then we recall how the `biblatex` package and `bib` module work. In Section 3, we show that some features of MIBib_TE_X⁵ may be of interest when we aim to process bibliographies for `biblatex` and ConT_EXt. Let us recall that MIBib_TE_X is a reimplement—written using the Scheme programming language (Kelsey et al., 1998)—of Bib_TE_X with particular focus on multilingual features. It was presented at the G_LT_EX 2004 meeting (Huffer, 2005). Section 4 introduces our programs `mlbiblatex` and `mlbibcontext`, they are variants of MIBib_TE_X, suitable for `biblatex` and ConT_EXt respectively. Finally we discuss some technical points.

2. In fact, some slight extensions, built out of the source files of Bib_TE_X—e.g., Bib_TE_X8 (Mittelbach et al., 2004, § 13.1.1) and Bib_TE_Xu (Voß, 2011, § 4.3)—have been designed in order to deal with encodings. Those recognised by L^AT_EX are given in (Mittelbach et al., 2004, § 7.5.2).

3. An introduction to this package in Italian is (Pantieri, 2009).

4. **Practical Extraction and Report Language**. A good introduction to this language is (Wall et al., 2000).

5. **MultiLingual Bib_TE_X**.

1. This distinction between *entries* and *references* originates from MIBib_TE_X’s terminology. We follow it throughout this article.

```
@BOOK{pollotta2007,
  AUTHOR = {Nick Pollota},
  TITLE = {Neutron Force},
  SERIES = {Don Pendleton's \emph{Stony
    Man}},
  NUMBER = 89,
  PUBLISHER = {Gold Eagle},
  TOTALPAGES = 352,
  YEAR = 2007,
  MONTH = jun}
```

FIGURE 1: Example using BibTEX's format.

1 Subtleties of the .bib format

Many L^AT_EX end-users have a *huge* number of .bib files encoded in the format defined by BibTEX (Mittelbach et al., 2004, § 13.2). In addition, many organisations are making .bib files available on the Web, e.g., ACM⁶'s digital library,⁷ IEEE⁸ Xplore's digital library,⁹ or CiteSeerX.¹⁰ So a new bibliography processor designed to work in conjunction with L^AT_EX should be able to deal with this .bib format. At first glance, it is not very complicated, all the metadata of a bibliographical entry given using the syntax 'KEY = value', as you can see in Fig. 1. In reality, this format is more subtle. For example, values may be surrounded by double quotes:

```
TITLE = "Il nome della rosa"
```

in which case a double quote character used within such a value must be surrounded by braces:

```
TITLE = "Die Energiej{\\"a}ger"
```

Values may also be surrounded by braces:

```
TITLE = {Grande Jonction}
```

in which case a double quote character can be used alone within such a value:

```
TITLE = {Murcos Verm\"achtnis}
```

The syntax for person names—see (Hufferlen, 2006) for more details—is accurate for simple cases, but may be surprising in such a case:

```
AUTHOR = {Jean {Le Clerc de la Herverie}}
```

(removing the braces surrounding 'Le Clerc de la Herverie' causes 'Herverie' to be interpreted as the last name, 'Jean Le Clerc' as the first name, and 'de la' as a particle). An additional point comes from the widespread habit of inserting L^AT_EX commands inside value parts of BibTEX fields, like the following snippet:

6. Association for Computing Machinery.
7. See <http://dl.acm.org>.
8. Institute of Electrical and Electronics Engineers.
9. See <http://ieeexplore.ieee.org/Xplore>.
10. See <http://citeseerx.ist.psu.edu>.

```
\documentclass{article}

\usepackage[%
  bibstyle=numeric,citestyle=authoryear]{%
  biblatex}
\addbibresource{sm.bib}
%% You must put the ".bib" suffix.

\begin{document}

Do you know \parencite{pollotta2007}? This is
an interesting thriller written by
\citeauthor{pollotta2007} and came out in
\citeyear{pollotta2007}.

\printbibliography

\end{document}
```

FIGURE 2: Example using the biblatex package.

```
TITLE = {\em Babylon Babies}
```

That complicates the generation of Web output.¹¹ Moreover, the following declaration:

```
TITLE = {\emph{Cosmos Incorporated}}
```

yields a title's specification which would be correctly interpreted by L^AT_EX, but ConT_EXt would not recognise the \emph command.

In other words, it is quite easy to transform the syntax 'KEY = value' into '<KEY>value</KEY>' if we adopt XML¹²-like syntax, or '(KEY value)' if some Lisp¹³-like syntax is preferred. On the contrary, destructuring fields' values may be more complicated. That is why you can find many converters from .bib files into other formats, but at the first level. Roughly speaking, only few of these programs run the risk of analysing the contents of fields' values.

2 Deferring reference-formatting

2.1 The biblatex package

Let us consider the example of a L^AT_EX document given in Fig. 2 and citing the bibliographical entry given in Fig. 1. This document uses the biblatex package. As aforementioned, one bibliography style—biblatex—is associated with this package, so the \bibliographystyle command is not to be used. The \bibliography command, 'traditionally' used, has been replaced in the preamble by the \addbibresource command of biblatex (Lehman, 2011, § 3.5.1). Let us recall that

11. This example with the \em command may seem to be quite artificial. However, some L^AT_EX commands are sometimes used as *workarounds* inside values denoting person names. A good example is given in (Mittelbach et al., 2004, p. 767).

12. eXtensible Markup Language.

13. L^IST Processor.

```
\entry{pollotta2007}{book}{%
\name{author}{1}{}{%
  {}{Pollotta}{P.}{Nick}{N.}{-}{-}{-}}
\list{publisher}{1}{Gold Eagle}
\strng{namehash}{PN1}\strng{fullhash}{PN1}
\field{labelyear}{2007}\field{sortinit}{P}
\field{number}{89}
\field{series}{%
  Don Pendleton's \emph{Stony Man}}
\field{title}{Neutron Force}
\field{month}{06}\field{year}{2007}
\endentry
```

(The TOTALPAGES field does not appear here because it is not known by biblatex.)

FIGURE 3: Reference used by the biblatex package.

the `\bibliography` command serves two purposes when BiB_T_EX's 'standard' bibliography styles are used: it specifies both .bib files and the place where the bibliography is to be typeset. The new command `\addbibresource` just specifies a .bib file to be searched in order to build the bibliography, whereas inserting the 'References' section within the document is done by the `\printbibliography` command. The biblatex package provides as well commands to access the value of fields such as AUTHOR, TITLE, YEAR; for example, `\citeauthor` and `\citeyear` in Fig. 2 are rendered as follows:

Do you know (Pollotta 2007)? This is an interesting thriller written by Pollotta and came out in 2007.

The biblatex package allows citations to be controlled precisely: you may number them or put the author-date system (Mittelbach et al., 2004, § 12.3) into action. You can do that globally by means of the style option, or separate the management of 'References' sections and the citations throughout the document, by means of the options `bibstyle` and `citestyle`, as we do in Fig. 2.

Many extensions have been introduced by biblatex. Numerous additional fields are recognised, for example, SUBTITLE, for a work's subtitle, in addition to its title (Lehman, 2011, § 2.2). Likewise, the grammar defining the PAGES field has been refined. Additional entry types can be handled, for example, @BOOKINBOOK, for items originally published as a standalone book and reprinted in collected works of an author (Lehman, 2011, § 2.1). In addition, you can use the standard fields YEAR and MONTH, or replace them by the DATE field, which allows the specification of date ranges (Lehman, 2011, § 2.3.8):

DATE = {2012-10-27/2012-10-28}

The specification of fields recognised by biblatex use types: for example, AUTHOR is a name list, SUBTITLE and TITLE are literals. Some types are described

```
\usemodule[bib] % Not needed for MkIV.
\setupbibtex[database=sm]
\setuppublications[numbering=yes]

\starttext

Did you read \cite[pollotta2007]?

\placepublications

\stoptext
```

FIGURE 4: Citations and bibliographies in ConT_EXt.

by means of regular expressions, e.g., the *date* type. This type information appears in .bbl files suitable for biblatex, as shown in Fig. 3.

2.2 biblatex with biber

Roughly speaking, if you use the biblatex package in conjunction with BiB_T_EX, you go on using the latter for tasks it does not perform satisfactorily.¹⁴ That is particularly true when entries are sorted since BiB_T_EX's sort procedures have been designed to work only on 'pure' ASCII¹⁵ texts.¹⁶ In other words, accents and other diacritical signs are processed improperly. For this use case, biber becomes an interesting bibliography processor because it correctly sorts entries in encodings other than ASCII. It can be specified by the backend option of the biblatex package:

```
\usepackage[backend=biber]{biblatex}
```

This option causes a .bcf¹⁷ configuration file—using XML-like syntax—to be built. Let *f* be a file name without suffix, the command '`biber f`' is equivalent to '`biber f.bcf`'.

The biblatex package's sorting option, usable with this backend, allows a bibliography to be sorted w.r.t. a sorting scheme:

```
\usepackage[backend=biber,sorting=nyt]{%
  biblatex}
```

This predefined sorting scheme nyt uses three successive sort keys, based on authors or editors' Names, then Years of publication, and Titles. The `\DeclareSortingScheme` command (Lehman, 2011, § 4.5.1) allows the definition of a new sorting scheme.¹⁸ In fact, the tools related to the biblatex package use additional fields for sorting (Lehman, 2011, § 2.2.3): the first pass is controlled by the

14. ... although some points have been improved with BiB_T_EX8.

15. American Standard Code Information Interchange. 16. Let us recall that BiB_T_EX's first version came out in 1985.

17. Biblatex Control File.

18. Such a definition is to be put in a document's preamble.

```
\startpublication[k=pollotta2007,t=book,
a={{Pollota}},y=2007,n=1,s=Pol07]
\author[][Nick][N.]{Pollota}
\pubyear{2007}\title{Neutron Force}
\series{Don Pendleton's {\em Stony Man}}
\volume{89}\pubname{Gold Eagle}\month{6}
\stoppublication
```

(The TOTALPAGES is not known by ConT_EXt bib module, either.)

FIGURE 5: Reference used by ConT_EXt.

PRESORT field; by default, some fields only used for sorting—such as SORTNAME, SORTYEAR—take precedence over the corresponding fields for ‘actual’ information—that is, AUTHOR or EDITOR, YEAR.¹⁹

2.3 ConT_EXt’s bib module

Let us consider Fig. 4 as an example of a source text for ConT_EXt using a bibliographical reference. This reference, as it should be produced by a bibliography processor, is given in Fig. 5. The bib module (Hoekwater, 2001) can be used with ConT_EXt MkII (CON, 2012a), it has been reimplemented in ConT_EXt MkIV (LuaT_EX) by Hans Hagen (CON, 2012b).

3 What can MiBibT_EX provide?

As aforementioned, some operations related to the .bib format are quite difficult; MiBibT_EX does some easily, and this can be profitable when .bbl files are to be generated for biblatex or ConT_EXt. An important point is the check operations performed by MiBibT_EX, stricter than BibT_EX’s, what may be useful. Last but not at least, some syntactical enrichment ease .bib files’ syntax, and this may be of interest for biblatex users.

3.1 Destructuring

As mentioned in the introduction, MiBibT_EX has been first designed and developed as possible replacement for BibT_EX (Huffle, 2003). As part of this task, we put into action an analysis of the values associated with BibT_EX fields, as deeply as possible. We have precisely designed an internal format for bibliographical items. In fact, when MiBibT_EX’s parser processes a .bib file, we can consider that it builds an XML tree of this file. More precisely, this program builds expressions according to the sXML²⁰ format (Kiselyov, 2005). For example, Fig. 1’s entry is translated into the XML tree given in Fig. 6. We can see that the author’s name has been split into these components. Likewise, some basic commands of T_EX and

```
<book id="pollotta2007" from="sm.bib">
  <author>
    <name>
      <personname>
        <first>Nick</first>
        <last>Pollota</last>
      </personname>
    </name>
  </author>
  <title>Neutron Force</title>
  <publisher>Gold Eagle</publisher>
  <number>89</number>
  <series>
    Don Pendleton’s <emph>Stony Man</emph>
  </series>
  <totalpages>352</totalpages>
  <year>2007</year>
  <month><jun/></month>
</book>
```

(The from attribute of the book element is set to the base name of the .bib file originally containing this entry.)

FIGURE 6: Fig. 1’s example given using XML syntax.

L^AT_EX—e.g., \em or \emph—are recognised and replaced by XML tags.

3.2 Additional check

When BibT_EX users begin to run MiBibT_EX, the most surprising feature is that the latter performs a more precise analysis of .bib files. When a field name is not recognised, a warning message is emitted.²¹ That *modus operandi* may be viewed as an advantage: for example—let us recall that EDITOR is optional within an entry of type @INPROCEEDINGS—if you inadvertently typed ‘EDITORS = ...’ instead of ‘EDITOR_L = ...’ inside such an entry, MiBibT_EX will warn you whereas BibT_EX will silently ignore that field. This feature may also be viewed as a drawback: if you specify a MONTH field, the associated value *must* be a symbol among jan, feb, ..., dec. Otherwise, MiBibT_EX stops with an error message. This convention may appear as too restrictive, but MiBibT_EX can sort w.r.t. month names, whereas BibT_EX or biber do not. To perform such an operation, month names must be recognised. Likewise, when years are to be sorted, MiBibT_EX applies a numerical sort whereas BibT_EX and biber sort years as strings, so the value associated with a YEAR field must be an integer;²² otherwise, an error message is emitted. More precisely, the fields subject to additional check are:

- the standard fields AUTHOR, EDITOR, MONTH, PAGES, and YEAR;

19. This *modus operandi* is not specific to biber, it is implemented within BibT_EX’s biblatex bibliography style.

20. Scheme implementation of XML.

21. ... but this is just a warning message; the corresponding information is not lost.

22. Negative values, for years BCE, are allowed.

```

(<italian? "Monteverdi" "Monteverdi")           ==> #f
(<italian? "Monteverdi" "Monteverdi" (lambda () #f) < 'uppercase-1st) ==> #f ; Default values.
(<italian? "Monteverdi" "Monteverdi" (lambda () 'ok))      ==> ok ; Equal strings.
(<italian? "Monteverde" "Monteverdi")           ==> #t
(<italian? "Monteverde" "Monteverdi" (lambda () 'ok) >)    ==> #f ; Descending order.
(<italian? "Monteverdi" "MonteVerdi" (lambda () 'ok))      ==> #f
(<italian? "Monteverdi" "MonteVerdi" (lambda () 'ok) < #f) ==> ok ; Case-insensitive equality.
(<italian? "Monteverdi" "MonteVerdi" (lambda () 'ok) < 'lowercase-1st) ==> #t ; Lowercase letters
                                                    ; take precedence.

(<italian? "Monteverde" "Monteverdi"
 (lambda ()
  (<italian? "Giuseppe" "Claudio"
   (lambda () (<arithmetical? 1567 1643 (lambda () ...)))))) ==> #f

```

FIGURE 7: Order relations handled by MIBIBTeX.

- the field DAY, used by numerous styles;²³
- the fields GENDER and TOTALPAGES, used by some bibliography styles associated with the jurabib package, as mentioned in (Mittelbach et al., 2004, § 12.5.1);
- two special fields used by MIBIBTeX: the LANGUAGE field, giving an entry's language (Hufflen, 2003), and the LASTSORTKEY field, allowing us to sort entries whose all the other sort keys are equal (Hufflen, 2008).

3.3 Order relations

In (Hufflen, 2007), we showed how the lexicographic order relations handled by MIBIBTeX were built. These order relations—implemented by means of Scheme functions—are language-dependent. A very simple use of the <italian? order relation—for Italian words—to compare two strings is given by the first example of Fig. 7—‘#t’ (resp. ‘#f’) stands for the ‘true’ (resp. ‘false’) value in Scheme. In reality, these functions are more powerful since they use optional arguments—controlling the behaviour—in addition to the two strings to be compared:

- the third is a *thunk*²⁴ that is called if the two strings are equal;
- the fourth is < (resp. >) for an ascending (resp. a descending) order;
- the fifth is #f for a case-insensitive comparison, uppercase-1st (resp. lowercase-1st) if uppercase (resp. lowercase) letters take precedence when two strings are different only by the case.

Fig. 7's second example shows the default values of these three additional arguments. By default,

these functions implement *strict* order relations, that is, *irreflexive*, asymmetric, and transitive; as < for numbers. But if you would like such an order relation to be non-strict—that is, reflexive, asymmetric, and transitive—just bind the third argument to the thunk (lambda () #t). The sixth example shows that our <italian? function defaults to a case-sensitive relation in which uppercase letters take precedence over lowercase ones, the seventh example shows how to proceed if you would like lowercase letters to take precedence. Finally, the last example shows how the third argument can be used to *chain* order relations: the idea is to sort person w.r.t. last names, then first names, birth dates, and eventually other information.²⁵ This feature—sketched in (Hufflen, 2008, § 4)—makes possible the implementation of multiple levels of sorting. More details about these order relations are given in (Hufflen, 2012b).

A sort key may refer to *optional* information, e.g., the month of an entry, not required as an entry's year is. MIBIBTeX addresses such problems by means of functions encompassing default values when we are looking for optional information. For example, the Scheme function returning a month's rank—from 1 to 12—works as follows; the expression:

```
(<month-position T default-value)
```

returns the month's rank if the *T* form contains month information, *default-value* if this information is not supplied. Within a chronological sort, just bind the second argument—*default-value*—to 0 (resp. 13) if you would like the forms without this information to come before (resp. after) the forms where such value is defined.

23. For example, the styles ‘apa...’, used by the American Psychology Association.

24. A zero-argument function, w.r.t. Scheme's terminology.

25. The *arithmetical?* function, used within this example, is analogous to our order relations, in the sense that its third argument is called if the two numbers given as first two arguments are equal. Otherwise it behaves like <.

```
@BOOK{cussler2004,
  AUTHOR = {Clive Eric Cussler,
    abbr => C. with
    first => Paul,
    last => Kemprecos},
  TITLE = {White Death},
  SERIES = {Numa Files},
  PUBLISHER = {Penguin Books},
  YEAR = 2004}
```

FIGURE 8: Extensions recognised by MIBIBTEX.

3.4 Syntactical extensions

MIBIBTEX's syntactical extensions about multilingualism have been explained in detail in (Hutfien, 2003). Even if the biblatex package includes some multilingual features and work with the babel package (Mittelbach et al., 2004, Ch. 9), it cannot use these features presently.²⁶ On the contrary, our extensions for authors' and editors' names can be directly usable by the biblatex package. In addition to BIBTEX's conventions, *keywords* may be used to point to the four parts—*First*, *von*, *Last*, *Junior*—of a name, what may be very useful in such a case:

```
first => Jean,
last => Le Clerc de la Herverie
```

(the four keywords 'first =>', 'von =>', 'last =>', 'junior =>' are available, the order of appearance being irrelevant). The 'abbr =>' keyword may be used when a first name is not abbreviated according to the standard way—that is, retaining only the first letter, following by a period—as shown in Fig. 8 (removing this keyword would cause the first name to be abbreviated by 'C. E.'). Let us mention that mixed specifications are allowed: the second name is given according to BIBTEX's conventions, followed by the redefined abbreviation of the first name. If an organisation's name is used as an author or editor, you can use the keywords 'org =>' for the name as it must be typeset and 'sortingkey =>' for the key used for sorting:

```
org => \GuIT-2012,
sortingkey => GuIT 2012
```

It is well-known that co-authors are connected by means of the 'and' keyword. As shown in Fig. 8, MIBIBTEX also allows the specification of *collaborators*, by means of the 'with' keyword.

26. A connection is planned for a near future. That should be quite easy for ConTeXt, in the sense that all the languages are available *a priori*—you do not have to put all the languages you use throughout a text as options of a module like the babel package (Mittelbach et al., 2004, Ch. 9).

4 MIBIBTEX's variants

When the approach of biblatex and ConTeXt is used, a bibliography processor has just to provide successive structures whatever the bibliography style is, such a style is put into action by customising the command of L^AT_EX or ConTeXt producing the final bibliography. So the *program* building these structures does not need to be customised by end-users. We can use techniques of 'actual' programming in order to get more efficiency, rather than interpreting a language as BIBTEX does for bibliography styles. When we programmed MIBIBTEX's kernel, we succeeded in using techniques related to functional programming in Scheme, we got an efficient program by compiling our functions. We have wanted to extend these advantages to the complete process of generating references for biblatex and ConTeXt. That yields two bibliography processors out of MIBIBTEX's kernel, written entirely in Scheme: *mlbiblatex* and *mlbibcontext*.

4.1 The mlbiblatex program

The *mlbiblatex* program generates .bbl files suitable for the biblatex package. You can run it as follows—the .aux suffix can be omitted—:

```
mlbiblatex f[.aux] key-expr lg-code
```

where:

f.aux is the auxiliary file where the information about bibliographical keys and database files has been stored;

key-expr gives successive sort keys, according to the pattern (m | n | t | y)*, where 'm', 'n', 't', 'y' respectively stand for 'Month',²⁷ 'Name' (person name as an author or editor), 'Title', 'Year'; all the other signs are ignored; there is no default order relation:²⁸ if no sign is recognised, the list of bibliographical items is left unsorted;

lg-code is the language's code to be used for sorting strings when person names and titles of works are compared; available values are DE for German, EN for English, FR for French, IT for Italian, PO for Polish; there is no default value.

See Fig. 3 for example.

A switch mechanism implemented by means of Scheme functions allows us to recognise the extensions introduced by biblatex only when our parser is running in a kind of 'mlbiblatex mode'. For

27. ... an item without month information being ranked after an item with such.

28. The default order relation used by both BIBTEX and biber would be specified by ynt. Let us recall that by default, these two programs do not use any information about month during the sort step.

example, new entry types of `biblatex` are processed. The same mechanism allows `biblatex` users to specify dates either by means of a `DATE` field, or by a `YEAR` field and optionally by a `MONTH` field and a `DAY` field, whereas the `DATE` field is not allowed in `BibTEX` standard styles.

4.2 The `mlbibcontext` program

The `mlbibcontext` program generates `.bbl` files suitable for `ConTEXt`. The corresponding command line looks like `mlbiblatex`'s:

```
mlbibcontext f[.aux] key-expr lg-code
```

and `f.aux`, `key-expr`, `lg-code` have the same meaning than in § 4.1. See Fig. 5 for an example. By the way, we can notice that the `\emph` command used in Fig. 1 and not recognised by `ConTEXt` has been replaced by the `\em` command.

Here also, another switch mechanism considers a new `@CONTEXTPREAMBLE` directive when a `.bib` file is parsed by the `mlbibcontext` program. This directive aims to replace the ‘traditional’ `@PREAMBLE` directive, often used to put definitions of new `LATEX` commands (Mittelbach et al., 2004, § 13.2.4). In particular, the `@CONTEXTPREAMBLE` directive can be used to define in `ConTEXt` some missing `LATEX` commands used inside `.bib` files.

4.3 The distribution

`MiBibTEX`'s distribution is located at:

```
http://disc.univ-fcomte.fr/home/~jmhufflen/
texts/superreport/smlbibtex-1.3.tar.gz
```

To install it easily, compile the source files using the `bigloo` (Serrano, 2010) Scheme compiler; the installation procedure (Hufflen, 2012a) uses the commands `configure` (Vaughn et al., 2000) and `make` (Loukides and Oram, 1996) of the `GNU`²⁹ toolchain; more details are given in (Hufflen, 2012a, § 4.2). The complete distribution's version number is given ‘classically’, that is, by means of number sequences. Versions of particular variants are labelled by geographical names; the current versions of `mlbiblatex` and `mlbibcontext` are ‘Breskens versions’, since they have been demonstrated at first at the `EuroTEX` 2012 conference, at Breskens, The Netherlands.

5 Discussion and conclusion

An important point to take note of is that the successors of `BibTEX` have introduced many often incompatible extensions.³⁰ Moreover, their design principles are different: `biblatex` has introduced new

fields whereas `MiBibTEX` has extended the syntax of the existing ones. For example, `biblatex` uses a `COLLABORATOR` field for additional collaborators whereas `MiBibTEX` links them by means of the keyword ‘`\with`’, as mentioned in § 3.4 (see also Fig. 8). These two techniques have advantages and drawbacks, already discussed in (Hufflen, 2011, § 3). Anyway, we think that the `.bib` format should evolve to take into consideration how modern tools deal with it.

The programs `mlbiblatex` and `mlbibcontext` currently have rough interfaces that should be enhanced and perfected: for example, command lines only get access to ascending orders, whereas `MiBibTEX`'s kernel can provide descending orders, as mentioned in § 3.3. These programs are work in progress and in a state that allows us to experiment which interface could be a better fit for `biblatex` and `ConTEXt`. That could lead to a new backend for the `biblatex` package:

```
\usepackage[backend=mlbiblatex,...]{%
  biblatex}
```

—and something equivalent for `ConTEXt`—other options could allow accurate information to be passed to `MiBibTEX`.

A limitation of `MiBibTEX`'s current version is its capability to handle only two encodings, for input `.bib` files as well as output `.bbl` ones. More precisely, `.bib` files are supposed to be encoded in Latin 1. The characters not included in this encoding can be reached only by using `TEX` commands. About generated `.bbl` files, either `MiBibTEX` detects that the Latin 1 encoding is used by looking into the document's preamble,³¹ in which case this encoding is used for the `.bbl` file produced; otherwise, this `.bbl` file is a pure ASCII file, all the accented letters being specified by means of `TEX` commands. Such behaviour is due to Scheme. `MiBibTEX` has been written using this language's fifth revision (Kelsey et al., 1998), not Unicode-compliant. Most of Scheme interpreters can deal with Latin 1, some—not all—accept other encodings, but in a non-portable way. Besides, we want our functions to be able to work on as many Scheme interpreters as possible. A new revision of Scheme is in progress³² and will be Unicode-compliant, so a future version of `MiBibTEX` should be able to deal with other encodings such as Latin 2, UTF³³-8, UTF-16, etc. When we started to program `MiBibTEX`, we planned for a future version the use of Unicode, so we organised our functions in order for this update not to require too deep changes. To end up and be honest, we already succeeded in adapting

29. Recursive acronym: `GNU` is `Not UNIX`.

30. For example, some `BibTEX` bibliography styles use an additional field for the total number of a book's pages. Often this field is named `TOTALPAGES` (see Fig. 1)—e.g., within the `jurabib` bibliography style—but the tools related to `biblatex` know this information as `PAGETOTAL`.

31. `BibTEX` just reads `.aux` files and never reads a `.tex` file (Mittelbach et al., 2004, § 12.1.3), whereas `mlbibtex` may look into a document's preamble.

32. See the Web page <http://scheme-reports.org>.

33. Unicode Transformation Format.

MLBIB_{TEX} to applications other than those initially planned, so we face the future with confidence.

Acknowledgements

Thanks to Gianluca Pignalberi for the Italian translation of the abstract and keywords. I am also grateful to the anonymous referee whose valuable suggestions allowed me to improve many points.

References

- François Charette and Philip Kime. *biber. A Backend Bibliography Processor for biblatex. Version biber 0.9 (biblatex 1.6)*. <http://freefr.dl.sourceforge.net/project/biblatex-biber/biblatex-biber/development/documentation/biber.pdf>, August 2011.
- Bibliographies in MkII*. CONTEXTGARDEN, <http://wiki.contextgarden.net/Bibliography>, April 2012a.
- Bibliographies in MkIV*. CONTEXTGARDEN, http://wiki.contextgarden.net/Bibliography_mkiv, July 2012b.
- Hans Hagen. *ConTeXt, the Manual*. <http://www.pragma-ade.com/general/manuals/cont-enp.pdf>, November 2001.
- Taco Hoekwater. The bibliographic module for ConTeXt. In *EuroTEX 2001*, pages 61–73, Kerkrade (the Netherlands), September 2001.
- Jean-Michel Huffle. MLBIB_{TEX}'s version 1.3. *TUGboat*, 24(2):249–262, July 2003.
- Jean-Michel Huffle. MLBIB_{TEX}: a survey. In *Proc. GUIT Meeting*, pages 171–179, Pisa, Italy, October 2005.
- Jean-Michel Huffle. Names in BIB_{TEX} and MLBIB_{TEX}. *TUGboat*, 27(2):243–253, November 2006. TUG 2006 proceedings, Marrakesh, Morocco.
- Jean-Michel Huffle. Managing order relations in MLBIB_{TEX}. *TUGboat*, 29(1):101–108, 2007. EuroBach_{TEX} 2007 proceedings.
- Jean-Michel Huffle. Revisiting lexicographic order relations on person names. In *Proc. Bach_{TEX} 2008 Conference*, pages 82–90, April 2008.
- Jean-Michel Huffle. A comparative study of methods for bibliographies. *TUGboat*, 32(3):289–301, October 2011. Proc. TUG 2011 conference.
- Jean-Michel Huffle. MLBIB_{TEX} and the biblatex package. In Tomasz Przechlewski, Karl Berry, and Jerzy B. Ludwiczowski, editors, *Twenty Years After. Proc. Bach_{TEX} 2012 Conference*, pages 91–99, Bachotek, Poland, April 2012a.
- Jean-Michel Huffle. Gestion d'ordres lexicographiques multilingues avec xindy et MLBIB_{TEX}. À paraître dans les *Cahiers GUTenberg*, 2012b.
- Richard Kelsey, William D. Clinger, and Jonathan A. Rees, with Harold Abelson, Norman I. Adams iv, David H. Bartley, Gary Brooks, R. Kent Dybvig, Daniel P. Friedman, Robert Halstead, Chris Hanson, Christopher T. Haynes, Eugene Edmund Kohlbecker, Jr, Donald Oxley, Kent M. Pitman, Guillermo J. Rozas, Guy Lewis Steele, Jr, Gerald Jay Sussman, and Mitchell Wand. Revised⁵ report on the algorithmic language Scheme. *HOSC*, 11(1):7–105, August 1998.
- Oleg E. Kiselyov. *XML and Scheme*. <http://okmij.org/ftp/Scheme/xml.html>, September 2005.
- Philipp Lehman. *The biblatex Package. Programmable Bibliographies and Citations. Version 1.6*. <ftp://ftp.tex.ac.uk/archive/Archive%20directory/macros/latex/expt1/biblatex/doc/biblatex.pdf>, July 2011.
- Mike Loukides and Andy Oram. *Programming with GNU Software*. O'Reilly & Associates, Inc., December 1996.
- Frank Mittelbach and Michel Goossens, with Johannes Braams, David Carlisle, Chris A. Rowley, Christine Detig, and Joachim Schrod. *The L^AT_EX Companion*. Addison-Wesley Publishing Company, Reading, Massachusetts, 2 edition, August 2004.
- Lorenzo Pantieri. L'arte di gestire la bibliografia con biblatex. *ArsTeXnica*, 8:48–60, Ottobre 2009.
- Oren Patashnik. *BIB_{TEX}ing*. Part of the BIB_{TEX} distribution, February 1988a.
- Oren Patashnik. *Designing BIB_{TEX} Styles*. Part of the BIB_{TEX} distribution, February 1988b.
- Manuel Serrano. *Bigloo. A Practical Scheme Compiler. User Manual for Version 3.3b*, March 2010.
- Gary V. Vaughn, Ben Ellison, Tom Tromey, and Ian Lance Taylor. *GNU Autoconf, Automake, and Libtool*. Sams, October 2000.
- Herbert Voß. *Bibliografien mit L^AT_EX*. Lehmanns Media, Berlin, 2011.
- Larry Wall, Tom Christiansen, and Jon Orwant. *Programming Perl*. O'Reilly & Associates, Inc., 3 edition, July 2000.

▷ Jean-Michel Huffle
FEMTO-ST (UMR CNRS 6174) &
University of Franche-Comté,
16, route de Gray,
25030 BESANÇON CEDEX
FRANCE
jmhuffle at femto-st dot fr

La forma del testo umanistico: la classe `suftesi`

Ivan Valbusa*

Sommario

Avvalendosi anche di riflessioni di carattere filosofico, questo articolo analizza i principali aspetti della composizione dei testi umanistici, dalla scelta del carattere e del layout allo stile dei titoli e degli elementi del testo. Introduce inoltre gli strumenti per impaginare un testo usando la classe `suftesi`.

Abstract

This article analyzes the main aspects that arise when typesetting a humanistic text, from the choice of font and layout to the style of titles and text elements (even making use of philosophical reflections). It also introduces the tools to typeset by using the `suftesi` class.

1 Breve storia della classe `suftesi`

Nell'ormai lontano giugno 2010 tenni il primo corso `LATEX` per la Scuola di Dottorato in Scienze umane e filosofia dell'Università di Verona, oggi come allora diretta dalla sociologa Paola Di Nicola. Fu il primo corso `LATEX`, per quanto ne so, tenuto per gli umanisti veronesi. Presso il Dipartimento di Informatica, invece, il prof. Enrico Gregorio tiene da anni un corso analogo per gli studenti dei corsi di laurea scientifici (in particolare per i futuri matematici e informatici).¹

Durante il corso mi presi l'impegno di creare una classe appositamente pensata per gli utenti umanisti. È nata così la classe `suftesi` (VALBUSA,

*Quanto di buono si trova nel mio stile deve molto ai consigli del formidabile prof. Tommaso Gordini e dell'altrettanto precisa prof.ssa Cinzia Canteri, ai quali va un ringraziamento particolare. Del brutto sono invece responsabile in prima persona. Ringrazio anche il prof. Claudio Beccari per le sue preziose ed estese lezioni tipografiche. Tutti gli esempi contenuti in questo articolo sono composti con il nuovo font Junicode.

1. Il corso del 2010 era composto di quattro lezioni da due ore *non* accademiche e vi hanno partecipato complessivamente 14 dottorandi del primo, del secondo e del terzo anno. Nel 2011 è stato riproposto per la Facoltà di Scienze della Formazione, mutuato anche dalla Facoltà di Lettere. Composto da 18 ore di lezione, dava la possibilità di acquisire 3 crediti formativi. L'identico corso è stato tenuto anche l'anno successivo, questa volta per la Facoltà di Lettere e mutuato da Scienze della Formazione e anche Lingue e Letterature Straniere. Il corso del 2012 fa parte del Joint Project 2011 "Formal Style for PhD theses with `LATEX`", coordinato dal prof. Enrico Gregorio e tuttora in corso. Il progetto prevede l'istituzione di corsi su `LATEX` per tutte le scuole di dottorato dell'ambito umanistico. Finora si sono tenuti corsi per le scuole in Scienze umane e filosofia, Studi umanistici e Studi giuridici.

2012), le cui prime tre lettere stanno appunto per "scienze umane e filosofia".

Inizialmente aveva un intento minimale: doveva essere uno stile elementare, sobrio e senza fronzoli, adatto a comporre diverse tipologie di testi, dai saggi filosofici alle tesi in psicologia e sociologia. In questo mi ha guidato un principio di semplicità e funzionalità, nella ferma convinzione che il bello e il giusto non siano solo questioni di gusto. Come diceva Eric Gill, «il bello è ciò che risulta piacevole a vedersi, e sono piacevoli alla vista le cose che appaiono il più possibile perfette nel loro adattarsi alla funzione» (GILL, 1931, 12). È una prospettiva che si ritrova in molti altri designer, come il nostro Bruno Munari: «il designer cerca quindi di dare forma il più possibile coerente alla funzione dell'oggetto, forma che nasce direi quasi spontaneamente, suggerita dalla funzione» (MUNARI, 1966, 26). Ancora: «se la forma di un oggetto risulterà "bella" sarà merito della strutturazione logica e dell'esattezza nelle soluzioni delle varie componenti. Il "bello" è la conseguenza del giusto» (MUNARI, 1966, 31).

La prima versione della classe forniva pochissime opzioni, ma ben presto si è arricchita fino a comprendere decine di varianti grafiche, tra le quali quattro stili compatti adatti alla composizione di veri e propri libri.

1.1 Bello, molto bello, pure troppo

Il target principale di `LATEX` in Italia è il laureando. Chi si avvicina al mondo della tipografia digitale capisce subito che con questo strumento potrà comporre testi *belli*. Il problema che nasce è: quanto belli? e dove finisce il bello e inizia l'eccesso?

Lo scopo di ogni impaginato è quello di far sentire il lettore a proprio agio e pare che la scelta migliore sia quella che più ne incontra le abitudini. Non deve cioè essere una novità: «un libro ben progettato è riconoscibile in quanto tale solo a pochi eletti. [...] Anche dall'esterno un libro davvero bello non può rappresentare qualcosa di nuovo, deve invece configurarsi come semplice perfezione» (TSCHICHOLD, 1975, 10). Troppo semplice?

Sappiamo bene a quali tipi di impaginato è abituato il relatore medio in ambito umanistico. Questa abitudine è diventata ormai una forma *universale*: "foglio A4, Times New Roman, interlinea uno e mezzo" (margini *ad libitum*, a seconda di quanto si vuole allungare il discorso). *Et el resto valet un figo seco!* direbbe il dolcinaro Salvatore. Detto così fa rabbrivire tutti quelli che hanno la

fortuna di avere ancora un po' di gusto tipografico, ma tant'è.

Con LATEX si hanno a disposizione molte classi di documento (più di quattrocento solo in TEX Live), ma la maggior parte di esse non si discosta molto dall'idea generale delle classi standard. Molte usano il nero o caratteri senza grazie per i titoli di sezione e hanno gabbie di testo molto ampie, perché sono pensate per la stampa su foglio A4. Non è quindi poi così strano che l'autore umanista venga attratto da stili più *umani*, come ClassicThesis (MIEDE, 2012).

Lo stile di André Miede è sicuramente molto bello. Si ispira a canoni assodati e ha come mentore d'eccezione Robert Bringhurst, uno dei guru della tipografia mondiale. Forse, però, è *troppo* bello per una tesi di laurea: non solo, infatti, non lascia indifferente il lettore, ma lo lascia troppo poco indifferente.

La classe `suftesi` nasce quindi con lo scopo di offrire uno stile (si spera) bello, ma non eccessivamente; che possa risultare gradevole a una fetta molto ampia di possibili lettori, ma allo stesso tempo non appariscente come ClassicThesis, senza perciò correre il rischio che il vestito della pagina, invece di valorizzare il contenuto grazie al proprio equilibrio, lo soffochi per l'eccessiva presenza. Il progettista di libri, infatti, «deve creare una presentazione congegnata in modo tale che la forma non metta mai in ombra il contenuto» (TSCHICHOLD, 1975, 8).

Questo discorso vale a maggior ragione nel campo delle scienze umane, dove l'abitudine a tesi ben impaginate è lontana dall'essersi sedimentata. I matematici e gli informatici, invece, hanno questo *habitus* ormai da anni, e ciò dovrebbe suonare perlomeno singolare ai cultori delle *belle* lettere.

Vediamo quindi quali sono le caratteristiche di questo stile, ricordandoci però sempre che il designer rimane «un subdolo cospiratore» (FLUSSER, 1990, 1), anche se spesso fa ciò a fin di bene.

2 La classe `suftesi`

Tutte le opzioni della classe `suftesi` sono del tipo $\langle \text{chiave} \rangle = \langle \text{valore} \rangle$ e sono state implementate usando il pacchetto `xkeyval`, che ha reso parecchio più semplice la gestione delle molte combinazioni. Alcune opzioni booleane potevano essere definite in maniera più leggera e senza ricorrere al pacchetto di Hendri Adriaens, visto che uno dei due valori è quello di default, ma ho preferito l'uniformità, che a livello utente dovrebbe essere apprezzabile. Inoltre, in questo modo sarà più facile, nelle future versioni, implementare nuove opzioni senza stravolgere l'impianto generale.

Nelle prossime sezioni vedremo quali sono le caratteristiche della classe e mostreremo come la si possa personalizzare a seconda delle proprie esigenze.

2.1 Forma della pagina

2.1.1 Strutture di documento

La classe prevede due *strutture* di documento: `book` e `article`. La struttura `book` è pensata per la composizione di testi suddivisi in capitoli ed è quella che viene caricata di default. Le due righe seguenti, pertanto, sono del tutto equivalenti:

```
\documentclass{suftesi}
\documentclass[
  documentstructure=book]{suftesi}
```

La struttura `article` è dedicata invece alla composizione di articoli, tesi triennali, dispense, rapporti tecnici, ecc., che per la loro mole non richiedono una suddivisione in capitoli. Si carica con:

```
\documentclass[
  documentstructure=article]{suftesi}
```

Visto che in un articolo non dovrebbero essere presenti capitoli, se nel documento con struttura `article` è presente un comando `\chapter` verrà emesso un errore e la composizione si bloccherà. La scelta della struttura è quindi, in una certa misura, vincolante.

Quando è attiva questa opzione vengono ridefiniti i comandi `\thesection`, `\thesubsection`, ecc., che altrimenti stamperebbero il numero di capitolo (ovvero 0), e i comandi di sezionamento che generano l'indice generale e quelli di tabelle e figure. Anche tutti i comandi di `biblatex` che generano sezioni sono stati opportunamente ridefiniti. In questo modo, per esempio, `\printbibliography` e `\printshortands` stampano un capitolo se si usa la struttura `book` e una sezione se si usa la struttura `article`. Analogamente per le relative opzioni: `subbibliography`, per esempio, produce una sezione con la struttura `book` e una sottosezione con la struttura `article`.

2.1.2 Font

In tipografia il font è l'elemento che più caratterizza la pagina: *tipografia* significa letteralmente “disegno del carattere”. Ogni font ha il proprio luogo d'elezione e ogni testo viene valorizzato solo da alcuni font ma, come sostengono molti esperti, vanno sempre preferiti quelli che più incontrano le abitudini del lettore perché, ricordiamolo, scopo principale della tipografia è facilitare la lettura: «il tipografo deve essere molto attento alla scelta del carattere, rendendosi conto che quanto più spesso andrà ad utilizzarlo, tanto più il suo disegno dovrà avvicinarsi all'idea generale di carattere tipografico che il lettore ha già in mente, e che risulta per forza influenzata dalle riviste, dai quotidiani e dai libri a lui più famigliari» (MORISON, 1967, 19).

Le scienze umane comprendono discipline anche molto diverse tra loro. Si va dalla filosofia del linguaggio e dalla psicologia sperimentale, che talvolta usano massicciamente il simbolismo logico-matematico, fino alle discipline storiche, passando

per la sociologia, la pedagogia e la letteratura in senso stretto. Qual è il font più familiare alle scienze umane? Generalmente il lettore medio di queste discipline è abituato a font umanistici (Garamond e simili), neumanistici (Palatino) e transizionali (Baskerville). Da qualche anno, purtroppo, ci si sta abituando al Times New Roman, che viene (ab)usato anche in pagine con gabbia molto larga e da editori non improvvisati. Il font più familiare è forse il Simoncini Garamond.

Come è noto, in TEX Live non è disponibile di default nessun Garamond e ritengo che questa sia una mancanza seria, oltre che una delle prove più tangibili del fatto che gli umanisti non sono ancora entrati veramente nel mondo LATEX. Nella prossima sezione vedremo come ovviare a questa mancanza.

Preso atto di questa lacuna, la scelta del carattere di default è caduta sul Palatino di Hermann Zapf, per tre ragioni: 1) ha un disegno familiare agli umanisti “puri”; 2) grazie al pacchetto `mathpazo` ha un buon supporto per la matematica, che può risultare utile agli umanisti “misti”; 3) è provvisto di vero maiuscoletto, che lo qualifica senza dubbio come font “serio”. Di fatto è uno dei font più trasversali attualmente disponibili su LATEX. Certo è un po’ ingombrante rispetto ad altri caratteri e per questo consiglio di usarlo a non più di 10 pt, soprattutto con i layout compatti che descriveremo a breve.

Se si compone con LATEX, quindi, `suftesi` carica per il testo normale il font Palatino con vero maiuscoletto e numeri minuscoli, per il testo a larghezza fissa il font Bera Mono opportunamente scalato e per il testo senza grazie l’Iwona di Janusz M. Nowacki.² Ecco il codice usato nella classe:

```
\RequirePackage[osf,sc]{mathpazo}
\RequirePackage[scaled=0.8]{beramono}
\renewcommand{\sfdefault}{iwona}
```

Inutile precisare che si possono usare anche gli altri font presenti nella distribuzione, per i quali esistono pacchetti dedicati come `fourier`, `lmodern`, `kpfonts`, soltanto per fare alcuni esempi. In questo caso si ricordi soltanto di richiamare l’opzione:

`defaultfont = standard`

che riporta il sistema al font di default.

Se si compone con XYLATEX, al contrario, non viene caricato alcun font, per non limitare la libertà dell’utente. In questo caso si consiglia un nuovo carattere genuinamente umanista, lo Junicon, di cui diremo qualcosa più avanti.

Alfabeto greco

Nella maggior parte delle scienze umane non capita quasi mai di dover scrivere formule matematiche.

2. Questa scelta si ispira all’*Arte di scrivere con LATEX* (PANTIERI e GORDINI, 2012).

(Forse è questo il motivo per cui LATEX è ancora poco diffuso in quest’ambito.) In compenso capita spesso di dover riportare passi in greco antico o moderno. Nelle facoltà di lingue, poi, non è raro dover scrivere in alfabeti ancora più inconsueti, come quello cirillico (GREGORIO, 2010). La scelta del carattere adatto a una classe dedicata alle scienze umane non può quindi prescindere dalla ricerca di un font greco bene assortito.

Il Palatino caricato attraverso il pacchetto `mathpazo`, purtroppo, non ha il font greco, pertanto si è rivelato subito necessario trovare un carattere che vi si adattasse bene. Ne ho individuati due: l’Artemisia e il Porson, entrambi della Greek Font Society.³ Il carattere Artemisia, con le sue grazie affilate, si integra particolarmente bene con il disegno del Palatino e per questo viene caricato di default. Al contrario, l’accoppiamento con l’elegante CBGreek, nato per essere accostato a un romano neoclassico, non è a mio avviso dei migliori. La figura 1 mette questi tre font greci a confronto.

Per cambiare il font greco è disponibile l’opzione:

`greekfont = artemisia, porson, cbgreek`

LATEX e i Garamond

Per gli amanti dei font Garamond, esistono almeno tre possibilità che ora descriveremo brevemente.

L’URW Garamond della scomparsa Unternehmensberatung Karow Rubow Weber non è disponibile di default né su TEX Live né su MIKTEX, a causa di questioni legate alla licenza di distribuzione, ma si può installare molto facilmente grazie allo script `getnonfreefonts`.⁴ Il font si carica con:

```
\usepackage[garamond]{mathdesign}
```

Sfortunatamente, il font che si ha in questo modo è tutt’altro che completo. Mancano infatti vero maiuscoletto e alfabeto greco, per esempio, ma in compenso il supporto per la matematica è buono. Il falso maiuscoletto di cui è dotato, peraltro, è decisamente scadente e se ne sconsiglia l’uso.⁵ Per l’alfabeto greco, invece, si può facilmente ovviare, utilizzando i font Artemisia o Porson, che si adattano bene anche al disegno di questo carattere.

Con XYLATEX sono facilmente utilizzabili tutti i font installati sul sistema. Tra i progetti più interessanti vanno ricordati il font Junicon, sviluppato da Peter S. Baker e disponibile su TEX Live, e l’EB Garamond di Georg Duffner.⁶ Mi soffermo solo sul primo, che a oggi si trova a uno stadio di sviluppo più avanzato.

3. Il font Porson è stato opportunamente scalato. Ringrazio Enrico Gregorio per aver fornito il codice.

4. Lo script può essere scaricato dal sito <http://www.tug.org/fonts/getnonfreefonts/>, dove si trovano anche le indicazioni per l’installazione.

5. Chi volesse il vero maiuscoletto lo dovrebbe acquistare e quindi caricare il pacchetto con l’opzione `expert`.

6. Quest’ultimo può essere scaricato dal sito <http://www.georgduffner.at/ebgaramond/>.

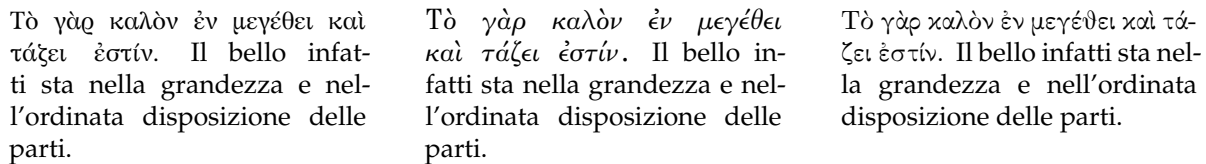


FIGURA 1: I tre font greci di *suftesi* a confronto. Da sinistra: Artemisia, Porson e CBGreeK. Il testo citato è tratto dalla *Poetica* di Aristotele, 7, 1450 b 35 (ARISTOTELE, sd).

Sebbene non abbia il supporto per la matematica, lo Junicode è un font molto completo. Per esempio ha il (vero) maiuscoletto, le *old style figures*, molte legature anche inconsuete, il supporto per l'alfabeto fonetico internazionale (IPA), un carattere greco molto raffinato e tutti i simboli necessari per scrivere nelle antiche lingue europee. Comprende pure degli ornamenti, che faranno sicuramente piacere ai nostalgici della tipografia d'un tempo.

Attualmente è la scelta elettiva per ottenere il miglior risultato con la classe *suftesi*. Basta caricarlo tramite il pacchetto `fontspec` e comporre con $\LaTeX$:

```
\usepackage{fontspec}
\defaultfontfeatures{Ligatures=TeX}
\setmainfont[Numbers=OldStyle]{Junicode}
```

Matematica e font umanistici

Il “problema” dei font OpenType o TrueType è che spesso non hanno un (buon) supporto per la matematica. Questo non sarebbe un problema se gli umanisti fossero solo letterati e storici. Ma si deve tener conto di tutte quelle discipline *miste* ricordate poc'anzi. La soluzione consiste nel prendere a prestito da un altro font i simboli necessari per comporre la matematica. Vediamo quindi quali sono le strade percorribili per comporre in Junicode un testo che contenga formule matematiche.

La prima soluzione prevede di usare il pacchetto `mathdesign`, che avendo il supporto matematico per l'URW Garamond si adatta benissimo anche allo Junicode:

```
\usepackage[garamond]{mathdesign}
\usepackage{mathspec}
\setmainfont[Numbers=OldStyle]{Junicode}
\setmathsfon(Digits, Latin, Greek){Junicode}
```

La seconda soluzione si affida al pacchetto `unicode-math`, ancora in fase di sviluppo, che mette a disposizione tre font matematici OpenType: Latin Modern Math, Asana Math e XITS Math:

```
\usepackage{unicode-math}
\setmathfont{Asana-Math.otf}
\setmainfont[Numbers=OldStyle]{Junicode}
\setmathsfon(Digits, Latin, Greek){Junicode}
```

Gli effetti finali sono molto simili. Numeri e lettere sono sempre Junicode, mentre i simboli matematici sono quelli di `mathdesign` nel primo caso, quelli del carattere Asana Math nel secondo.

Entrambe le soluzioni non sono soddisfacenti per testi matematici *veri*, perché in alcune situazioni si hanno spaziature decisamente sbagliate. Tuttavia rappresentano un buon compromesso in testi che contengano una matematica elementare ovvero poche formule, che eventualmente si possono sistemare a mano.

Una terza soluzione consiste nell'usare un font diverso per la matematica. Così facendo, però, non vi sarà coerenza tipografica tra testo in modalità matematica e testo normale:

```
\usepackage{fontspec}
\setmainfont[Numbers=OldStyle]{Junicode}
\usepackage{eulervm}
```

In questo modo si avrà il testo in modalità matematica composto con i font Euler di H. Zapf, i numeri in Junicode, e i simboli matematici mancanti in Computer Modern.

2.1.3 Layout

Ciò che accomuna tutte le tesi è che sembrano tutte delle *tesi*, sebbene a volte non vi si sostenga alcuna tesi. Questo pleonasma voluto descrive meglio di mille parole l'abisso che dal punto di vista tipografico separa le tesi dai libri e dagli articoli su cui tutte le tesi si basano: ci formiamo su testi generalmente belli (anche se a volte cattivi) ma leggiamo tesi quasi sempre brutte (anche se spesso buone).

Si arriva poi al paradosso per cui le tesi vengono rifinite con rilegature costose, magari con il testo dorato sul frontespizio. Tecnicamente pregevoli, ma tipograficamente demenziali. Chiunque abbia dei genitori con i capelli grigi sa che è molto meglio vestirsi con abiti poveri ed essere ben puliti, che incravattarsi per nascondere la sporcizia. Allo stesso modo, una tesi in (simil)pelle è ridicola, se aprendola vi ritroviamo la famigerata forma universale ricordata sopra.

Il formato adatto a un testo dipende da molti fattori, anche storici, perché no, ma *in primis* dal target e dal potenziale mercato. Per le tesi quest'ultimo è una variabile trascurabile, quindi ciò che conta è il target, ovvero il docente/ricercatore universitario.

(Quasi) nessun volume di saggistica, a cui è generalmente abituato l'umanista, è in formato A4, quello più diffuso per le tesi di laurea (più o meno rifilato). Gli unici volumi in formati così generosi (e anche maggiori) sono generalmente testi

TABELLA 1: Dimensioni e proporzioni dei cinque layout di pagina di *suftesi*. Tranne quelle relative a *elements*, date in punti tipografici, le dimensioni di testo e pagina sono espresse in millimetri.

	Opzioni				
	standard	periodical	compact	supercompact	elements
Larghezza testo	110	110	110	100	251
Altezza testo	220	165	165	150	502
Larghezza pagina	210	170	160	140	374
Altezza pagina	297	240	240	210	648
	Proporzioni				
	standard	periodical	compact	supercompact	elements
Margini alto/basso	1/2	2/3	2/3	2/3	1/2
Margini interno/esterno	1/2	2/3	2/3	2/3	2/5
Gabbia del testo	1/2	2/3	2/3	2/3	1/2
Pagina	1/√2	1/√2	2/3	2/3	1/√3

molto pesanti di arti grafiche. Ora, una tesi è molto più vicina a un saggio che a un catalogo d'arte e, fatte salve quelle che richiedono ampi spazi per accogliervi le immagini (si pensi a lavori di storia dell'arte), si possono scrivere benissimo tutte le altre in un formato più piccolo.

A partire dalla versione 0.5 ho così deciso di portare avanti una battaglia culturale per mettere in discussione e possibilmente cambiare l'usanza di rilegare le tesi in formato A4. Ho quindi curiosato tra le tesi consegnate nella Facoltà di Lettere della mia università e ho scoperto una cosa molto interessante: nessuna è in formato A4. Le differenze dovute al rifilo sono a volte anche notevoli e tanto è bastato per capire che il tacito accordo sul margine lasciato al rilegatore è talmente ampio, che una tesi in formato molto più ridotto non avrebbe potuto essere messa in discussione. In caso di controversie, il tutto potrebbe essere risolto attraverso il paradosso del sorite (sempre che le norme redazionali della facoltà non usino argomenti *fuzzy*).⁷

Ho pensato quindi di mettere a disposizione, oltre al classico formato A4, altri layout che risultassero più idonei a una tesi e che permettessero di impaginare anche veri e propri libri. La scelta è caduta inizialmente su uno snello 16 cm × 24 cm, a cui si è affiancato il più diffuso 17 cm × 24 cm.⁸ Completa il set un piccolo 14 cm × 21 cm per chi volesse comporre un vero e proprio "tascabile" e uno slanciato 1 : √3 che vedremo ampiamente nel-

la sezione 3.2. Le specifiche di questi layout sono sintetizzate nella tabella 1.

I layout messi a disposizione dalla classe sono quindi in totale cinque e si possono selezionare con la seguente opzione:

```
pagelayout = standard, periodical,
             compact, supercompact,
             elements
```

standard Stampa una pagina in formato A4.

La gabbia del testo è in proporzione 1 : 2, così come i margini interno/esterno e alto/basso. L'accoppiamento di questa gabbia con una pagina in proporzione 1 : √2 si rifà a una raccolta di opere di Orazio ricordata da BRINGHURST (1992, 184), ma con margini interno/esterno in proporzione 1 : 2 anziché 1 : 4.

periodical Stampa su pagina A4 una pagina virtuale di 17 cm × 24 cm e mostra i crocini di taglio. Aggiungendo l'opzione `crop=false` si ottiene un PDF con le dimensioni reali. La gabbia del testo misura 11 cm × 16,5 cm ed è in proporzione 2 : 3. Anche le coppie di margini interno/esterno e alto/basso hanno la stessa proporzione.

compact Come **periodical** ma con una pagina di 16 cm × 24 cm. Gabbia del testo e proporzioni dei margini sono gli stessi dell'opzione precedente.

supercompact Come **periodical** ma con pagina in formato 14 cm × 21 cm e gabbia del testo di 10 cm × 15 cm. I margini interno/esterno e alto/basso sono sempre in proporzione 2 : 3, come la pagina.

elements Riproduce il layout di pagina di BRINGHURST (1992). La pagina è in proporzione 1 : √3 e la gabbia del testo in proporzione 1 : 2. Va precisato che quest'opzione riproduce *solo* il formato della pagina e della gabbia del testo di

7. Il paradosso del sorite (da *sōrōs*, "mucchio") noto anche come paradosso del calvo, è un argomento logico usato per mostrare la vaghezza di alcuni concetti. Se una persona ha 100 000 capelli non è calva. Ma se togliamo un capello non sarà comunque calva, perché un capello non fa differenza. Si procede così fino a dimostrare che chi ha un capello non è calvo. Ma ciò è falso, naturalmente.

8. Con il primo formato è stato composto SCHENA (2011), con il secondo MADINELLI (2012) e ANDREOLI (2012). Inizialmente le segreterie hanno fatto qualche resistenza. Poi qualcuno si è ricordato che in uno stato di diritto ciò che non è esplicitamente vietato è sicuramente lecito e quindi le tesi sono state accettate, anche se con una restrizione: la copia ufficiale, depositata presso la segreteria, doveva essere in formato A4.

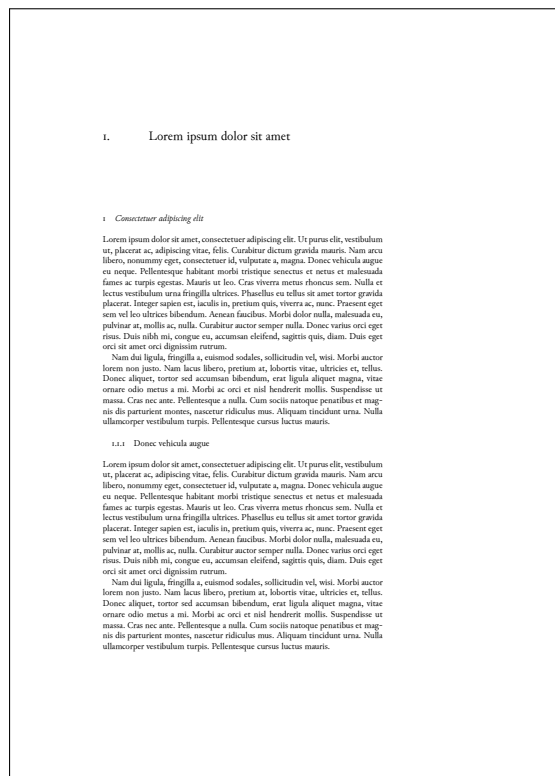


FIGURA 2: Lo stile di default nella prima versione della classe. Ora il punto dopo il numero di capitolo è stato omesso, ma può essere ripristinato con l'opzione `chapnumstyle=dotarabic`.

Bringhurst. Per riprodurre anche gli altri elementi esiste l'opzione `style=elements` (vedi sezione 3.2).

In tutti i layout il rientro della prima riga è impostato a 1 em. Per ripristinare il rientro standard di 1,5 em è sufficiente l'opzione:

```
parindent = wide
```

2.1.4 Titoli

Il primo progetto della classe prevedeva un unico stile per i titoli di capitolo, impostato anche adesso di default (con una leggera differenza) e mostrato nella figura 2.

Il titolo di capitolo è composto in tondo e corpo `\large` e si ispira a Tschichold (1975). È uno dei modi più elementari per realizzarlo. Da questa prima idea si sono poi sviluppate varianti sul tema, mantenendo sempre costante l'obiettivo di non cadere nel barocco.

Titolazioni così elementari e apparentemente banali, infatti, permettono di giocare liberamente su alcune variabili. La stessa pagina, infatti, mantiene il proprio aspetto gradevole cambiando semplicemente la posizione di titolo e numero. Nella figura 3 possiamo confrontare quattro varianti, ottenute con altrettante opzioni di `suftees`.

La classe mette a disposizione tre opzioni per variare lo stile di titoli e numeri delle sezioni. Le

prime due permettono di modificare rispettivamente carattere e stile dei titoli delle sezioni fino al quarto livello. Nelle due opzioni seguenti, `<level>` sta per `part`, `chap`, `sec`, `subsec` o `subsubsec`.

```
<level>font = roman, italic, smallcaps
```

Componi il titolo del `<level>` rispettivamente in caratteri romani, in corsivo e in maiuscolo. In quest'ultimo caso, naturalmente, il font in uso deve esserne provvisto. Quest'opzione non andrebbe quindi usata, per esempio, con l'URW Garamond installato secondo le indicazioni date sopra.

Le sezioni oltre il secondo livello non dovrebbero essere composte in maiuscolo se non in testi molto particolari nei quali, per esempio, si usassero le sottosezioni per inserti di approfondimento che meritano una titolazione più "evidente" delle altre.

```
<level>style = left, center, right,
               parleft, parcenter,
               parright
```

I primi tre valori stampano il titolo del `<level>` rispettivamente a sinistra, al centro e a destra; gli ultimi tre sono analoghi ai precedenti, ma stampano il titolo sotto il numero. Non si dovrebbe sentire la necessità di usare le opzioni `par-` per le sezioni oltre il secondo livello, ma non è escluso che per alcune opere ciò possa risultare funzionale.

La prossima opzione permette di personalizzare il numero della parte e delle sezioni di primo livello. Qui `<level>` sta quindi per `part`, `chap`, `sec`.

```
<level>numstyle = arabic, roman, Roman,
                  dotarabic, dotroman,
                  dotRoman
```

Le prime tre opzioni stampano il numero del `<level>` rispettivamente in cifre arabe, romane maiuscole e romane maiuscole; le altre tre aggiungono un punto dopo il numero.

La libertà di personalizzare la classe lasciata all'autore è notevole, tuttavia si è cercato di impedire il più possibile che si possano produrre brutture tipografiche. È però necessario qualche consiglio ulteriore.

Il punto dopo il numero, per esempio, ha senso solo in alcuni casi. Con la struttura `book` può essere usato sempre con un ottimo risultato per i numeri di capitolo, perché in questo caso il titolo ha un'evidente autonomia dal resto del testo. Con la struttura `article`, invece, il punto dopo il numero di sezione dovrebbe essere usato soltanto assieme alle opzioni `<level>style=parleft`, `parcenter`, `parright`, perché solo in questi casi il titolo ha una sufficiente indipendenza rispetto agli altri titoli di sezione. Inoltre sconsiglio di usare assieme le opzioni `<level>numstyle=dot-` e `<level>style=center`.

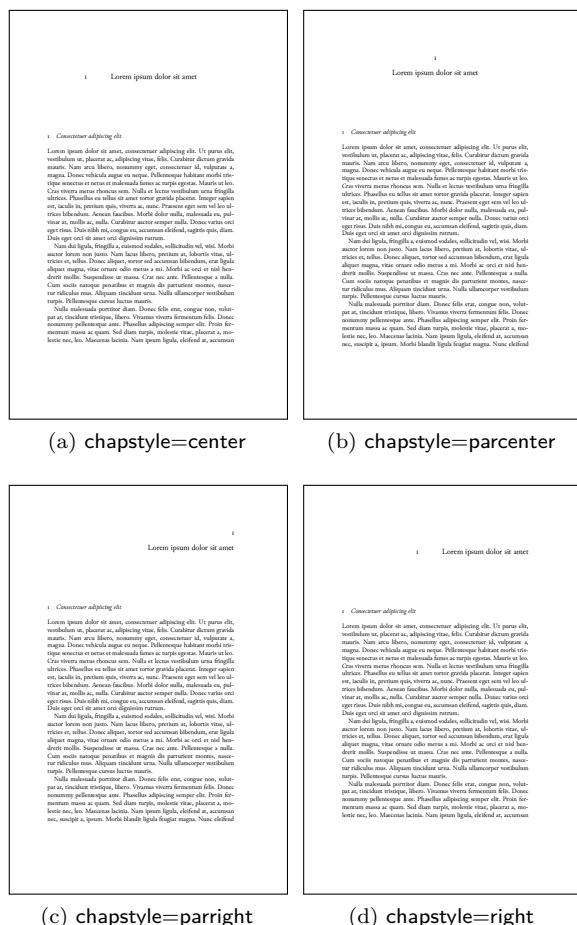


FIGURA 3: Quattro varianti dello stile di base mostrate sulla pagina compatta periodical (17 cm × 24 cm).

2.1.5 Stili di pagina

Una volta definiti layout e stili delle sezioni, rimane da impostare lo stile della pagina, caratterizzato dalla natura di testatine e piedini.

Le testatine hanno essenzialmente due funzioni. La prima è di ordine strumentale: in testi molto corposi ci informano sul titolo del capitolo o della sezione che stiamo leggendo. La seconda è di ordine estetico: la pagina che ne è provvista possiede una propria completezza formale e sembra infondere a molti un rassicurante senso di ordine a prescindere dalla loro reale funzionalità. In questo modo le testatine rivestono il ruolo di un vero e proprio accessorio tipografico, alla stregua della cravatta o della *pochette*. D'altra parte, l'occhio vuole sempre la sua parte.

Chiaramente, riportare il titolo dell'opera o il nome dell'autore nelle testatine non risponde ad alcun criterio funzionale, visto che simili informazioni dovrebbero essere date nel frontespizio o nella copertina. E in ogni caso non si dovrebbe sentire la necessità di essere informati di ciò in ogni pagina. Tale vezzo risponde ufficialmente a esigenze di copyright, anche se credo che il motivo estetico sia quello preponderante. In una tesi di laurea, tutta-

via, potrebbe essere utile per ricordare al lettore che la tesi è un prodotto dello studente, sebbene guidato e aiutato dal relatore.

In tutti gli stili di pagina della classe, testatine e piedini vengono composti nello stesso corpo delle note a piè di pagina e non prevedono linee divisorie. Esistono complessivamente sei stili di pagina, selezionabili con la seguente opzione:

```
headerstyle= inner, center, plain,
             authortitleinner,
             authortitlecenter, elements
```

inner Con la struttura book stampa l'espressione *Capitolo n* e il titolo del capitolo nella parte interna della testatina delle pagine pari e dispari, rispettivamente. Con la struttura article compare invece il titolo della sezione su entrambe le pagine. Il numero di pagina viene stampato nella parte esterna (figura 4a).

center Simile a inner ma con testatine centrate e numero di pagina al centro del piede (figura 4b).

plain Omette le testatine e stampa il numero di pagina al centro del piede.

authortitleinner Stampa nome dell'autore e titolo del lavoro (dichiarati con comandi `\author` e `\title`) nella parte interna della testatina delle pagine pari e dispari, rispettivamente. Il numero di pagina compare invece nella parte esterna.

authortitlecenter Simile all'opzione **authortitleinner** ma con testatine centrate e numero di pagina al centro del piede.

elements Invece delle consuete testatine stampa delle spalle correnti nel margine. Il titolo della sezione e del capitolo compaiono rispettivamente nel margine delle pagine pari e dispari, mentre il numero di pagina viene stampato nell'esterno del piede. È lo stile usato in BRINGHURST (1992), che vedremo nella sezione 3.2.

Lo stile del carattere delle testatine può essere modificato con l'opzione:

```
headerfont= roman, italic, smallcaps
```

Con questa opzione si possono comporre le testatine rispettivamente in carattere tondo, corsivo e maiuscolo. Non è attiva con lo stile **elements**.

2.2 Elementi del testo

2.2.1 Citazioni fuori corpo

Nelle scienze umane, si sa, si fanno molte citazioni, in corpo e fuori corpo. Ciò è dovuto al fatto che, a differenza delle scienze esatte, che trovano la loro giustificazione in sé stesse, nelle prime è

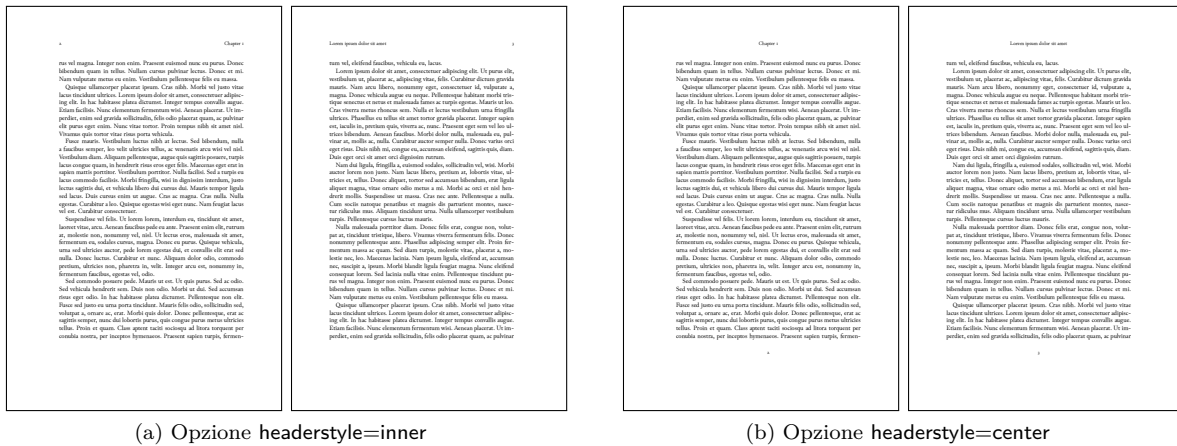


FIGURA 4: Due stili di pagina che si possono ottenere con la struttura book.

inevitabile il ricorso all'autorità. Questo accade frequentemente in tutte le discipline storico-filosofiche e in quelle giuridiche, ma è una caratteristica trasversale di tutti gli ambiti in cui non si possono usare numeri. Leibniz sognava di creare un calcolo che permettesse di risolvere ogni questione, anche metafisica, mettendosi a tavolino e facendo dei (semplici) calcoli. Ma si sbagliava.

Le cose stanno così e in questo luogo non prendo ulteriore posizione in merito. Le citazioni servono, e se usate con saggezza migliorano la qualità del testo. Specie in alcune discipline come la filosofia, è poi pacifico che il lettore, come l'autore, prova un certo piacere nel vedere una pagina con qualche citazione fuori corpo. Ciò agisce in lui come una sorta di assicurazione.

Ecco quindi che la forma delle citazioni fuori corpo diventa importante. Per soddisfare l'unico requisito che devono possedere, cioè potersi distinguere facilmente dal contesto, si possono percorrere diverse strade. Se le citazioni in corpo vengono semplicemente racchiuse tra virgolette (alte o basse), con le citazioni fuori corpo si può agire su tipo di carattere, corpo, spazi verticali e rientri.

Sono rari i testi che compongono le citazioni fuori corpo con un carattere diverso dal testo normale, mentre è parere di tutti che la spaziatura verticale prima e dopo la citazione sia una scelta imprescindibile, anche se non sufficiente. Ecco perché, di norma, sono composte anche in corpo minore e rientrate a destra e a sinistra.

Le classi standard impostano un rientro di 2,5em, maggiore del rientro della prima riga (1,5em) e compongono le citazioni nello stesso corpo del testo. La classe suftesi adotta una scelta più semplice e più popolare nelle scienze umane: il rientro è lo stesso di quello della prima riga del testo normale e il corpo è lo stesso di quello delle note a piè di pagina. Non si è agito invece sul rientro della prima riga del testo citato, che non dovrebbe *mai* esserci per il semplice fatto che non avrebbe alcuna funzione (o ragione).

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetur adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in

(a) Opzione quotestyle=center

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetur adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in

(b) Opzione quotestyle=right

FIGURA 5: I due stili di citazione fuori corpo previsti da suftesi. Entrambi possono essere ulteriormente personalizzati con l'opzione quotestyle=small.

La classe fornisce due opzioni per personalizzare le citazioni in display:

`quotestyle= center, right`

Fa rientrare la citazione rispettivamente a destra e a sinistra (figura 5a) o solo a sinistra (figura 5b) del valore di `\parindent`.

`quotesize= footnotesize, small`

Componi le citazioni rispettivamente in corpo `\footnotesize` e `\small`.

2.2.2 Note

Un altro elemento che caratterizza talune discipline umanistiche è la nota a piè di pagina. In ambiti



FIGURA 6: I quattro stili di note a piè di pagina disponibili con **suftesi**. Lo stile (a) è impostato di default.

come la filosofia e la storia le note sono spesso più importanti del testo. Generalmente sono molto lunghe e non è raro imbattersi in testi con note lunghissime spalmate su due e anche tre pagine.⁹

Tutto ciò viola il buon senso tipografico, ma l'abitudine è talmente sedimentata che volerla cambiare è una battaglia persa in partenza. Le note sono per taluni irrinunciabili e per altri diventano la *cifra* di un certo stile di scrittura. Una classe pensata per le discipline umanistiche non può quindi prescindere dall'offrire un certo numero di opzioni per saziare il desiderio di distinguersi in una parte così pregnante del saggio: il piè di pagina.

Nel testo l'indice di nota appare come di consueto ad esponente, mentre nel piè di pagina si può scegliere tra quattro stili diversi, il primo dei quali è quello di default (figura 6).

footnotestyle= bulged, hung, dotted, superscript

bulged Il numero è in linea con il testo della nota e sporge oltre il margine sinistro della gabbia. È lo stile adottato nell'unica nota a piè di pagina presente in BRINGHURST (1992, 76).

hung Il testo della nota è rientrato a partire dalla seconda riga. È identico al precedente, ma con i numeri di nota allineati al margine sinistro della gabbia.

9. Chi sentisse il desiderio di vedere questo orrore tipografico può sfogliare VALBUSA (2008).

dotted L'indice di nota è in linea con il testo della nota e seguito da un punto.

superscript L'indice di nota è a esponente.

Se un testo contiene poche note, come è auspicabile, nasce il seguente dilemma: abbiamo ancora bisogno di una numerazione? Se vogliamo riferirci a una nota in una parte qualsiasi del testo, probabilmente sì. Se però non abbiamo questa esigenza e si hanno pochissime note per pagina, diciamo al massimo tre, la numerazione si può ritenere superflua. Ecco quindi che è stata creata un'opzione per generare note con marcatori non numerici:

fewfootnotes= true

Quest'opzione è utilizzabile *solo se* nel documento ci sono al massimo tre note per pagina (in caso contrario si ha un'errore e la composizione si blocca). La sequenza dei simboli usati come marcatori è *, **, *** e ricomincia a ogni pagina.

Esiste poi un comando per produrre una nota con un simbolo qualsiasi come marcatore:

\xfootnote[*simbolo*]{*testo della nota*}

Il comando non incrementa il contatore della nota, quindi può essere usato in qualsiasi punto del documento senza modificare la numerazione delle altre note eventualmente presenti.¹⁰

10. L'idea di queste due funzionalità è di Tommaso Gordini.

Un ultimo doveroso cenno riguarda le note a margine, sebbene non siano un elemento indispensabile della pagina. In `suftesi` vengono composte a bandiera, in corsivo e nello stesso corpo delle note a piè di pagina. Un'opzione permette di disabilitarle:

```
marginpar = false
```

2.2.3 Liste

Le liste sono strutture molto particolari, diffuse soltanto in pochi ambiti delle scienze umane. In filosofia, per esempio, sono spesso deprecate perché rendono il testo troppo tecnico (e forse anche troppo chiaro). In altri contesti, come la sociologia o la psicologia, invece, sono molto utilizzate.

Le liste della classe standard `book` sono pensate per una gabbia di testo ampia e il loro trattamento grafico è funzionale: sono ben visibili, con un generoso rientro a sinistra e a destra e le voci sono separate da un ampio spazio verticale.

In un testo umanistico (e non solo) si deve tener conto anche di questo aspetto: liste troppo evidenti spezzano l'unità formale del testo, che dipende anche dalla sua immagine generale. In altre parole, le liste ci possono essere ma non devono sembrare *troppo* delle liste. Ecco perché sono state ridefinite, riducendo innanzitutto lo spazio verticale tra le voci a 0,5ex ed eliminando lo spazio tra i capoversi interni. Il punto usato nell'ambiente `itemize`, poi, è in un più sobrio grigio.

Eric Gill fa un'interessante distinzione tra leggibilità e agio nella lettura: «la leggibilità sembra sia verificabile attraverso test della visione e una spiegazione razionale, e ciò può essere vero, ma l'agio nella lettura è evidentemente una questione molto più sfumata, che coinvolge considerazioni legate al più ampio contesto degli amori e degli odi umani» (GILL, 1931, 86). Il caso delle liste esemplifica proprio quanto detto da Gill. Esse, infatti, sono state definite tenendo conto più delle passioni umane che della loro funzione. D'altra parte anche il cuore vuole sempre la sua parte.

Per quanto riguarda il rientro, si è tenuto conto della gabbia del testo, che è abbastanza stretta a prescindere dal layout usato. Lo spazio, quindi, è quanto mai prezioso. Anche per le liste si è seguito il consiglio di Bringhurst: di default i numeri o i punti dell'elenco sono esposti nel margine sinistro (figura 7a). In questo modo il primo livello di ogni lista ha lo stesso spazio del testo normale, ma nel contempo, grazie al numero sporgente, svolge egregiamente il proprio ruolo di elenco.

Questo stile è sicuramente insolito, e non solo per le scienze umane. È forse uno degli aspetti più azzardati della classe, assieme all'equivalente stile di nota a piè di pagina visto sopra. Per questo motivo si è definita un'opzione per personalizzare le liste:

```
liststyle = bulged, aligned, indented
```

`bulged` I marcatori degli elenchi sporgono nel margine sinistro della pagina.

`aligned` I marcatori degli elenchi sono allineati con il margine sinistro della gabbia.

`indented` Le liste sono rientrate del valore di `\parindent`.

Come si sarà notato negli esempi riportati, la classe non usa il nero (troppo scientifico). Anche le etichette delle descrizioni sono state quindi modificate, optando per lo stile corsivo.¹¹

2.2.4 Didascalie

Le didascalie degli oggetti mobili sono composte con carattere normale in corpo `\small` e terminano automaticamente con un punto fermo. A differenza di quanto avviene con le classi standard, nel caso in cui la didascalia sia di una sola riga, viene allineata ugualmente a sinistra. Vi è in tutto un altro stile oltre quello di default, che si differenzia dal primo solo per l'uso di font senza grazie:

```
captionstyle = elements
```

2.2.5 Indice generale

Inizialmente la classe proponeva un unico stile di indice generale, nel quale le pagine relative a ogni voce compaiono alla fine della voce stessa, seguite da uno spazio di 1em (figura 8a). Alla base di questa scelta c'è un principio di funzionalità: l'indice serve per trovare semplicemente le pagine a cui ciascuna voce si riferisce e questa forma risponde in pieno a tale esigenza. È il tipo d'indice usato in BRINGHURST (1992). Molti utenti, tuttavia, rimangono sorpresi dal suo effetto grafico "scomposto" e la spiegazione data sopra non sempre risulta convincente. (È una delle occasioni in cui si nota chiaramente che la *forma* grafica vince sulla *funzione*.)

A partire dalla versione 1.0, la classe mette a disposizione un nuovo stile di indice generale, ispirato a diverse pubblicazioni italiane (per esempio LESINA (2009)). In questo tipo di indice i numeri di pagina sono incolonnati a sinistra anziché a destra (figura 8b), ottenendo così due vantaggi: da una parte si assolve al principio in base al quale i numeri di pagina devono essere vicini al proprio riferimento, dall'altra si evita l'effetto di movimento creato dall'indice in stile *ragged*, che non a tutti piace.

Oltre a questo è possibile avere anche un classico indice con numeri di pagina allineati al margine destro, con o senza linee puntate tra le voci e i numeri di pagina. L'opzione che permette di scegliere lo stile dell'indice generale è la seguente:

11. Chi preferisse uno stile diverso, per esempio un carattere senza grazie, può ricorrere ai comandi del pacchetto `enumitem`, già caricato dalla classe con l'opzione `inline`, che mette a disposizione anche gli ambienti `itemize*`, `enumerate*` e `description*`, utili per comporre liste orizzontali.

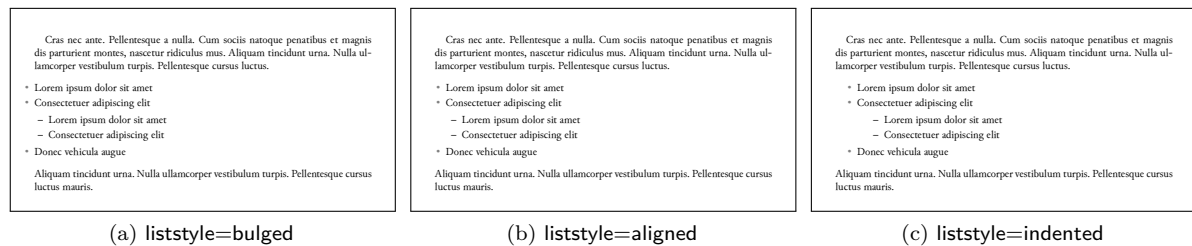


FIGURA 7: I tre stili di liste disponibili con suftesi. Il primo è quello di default.

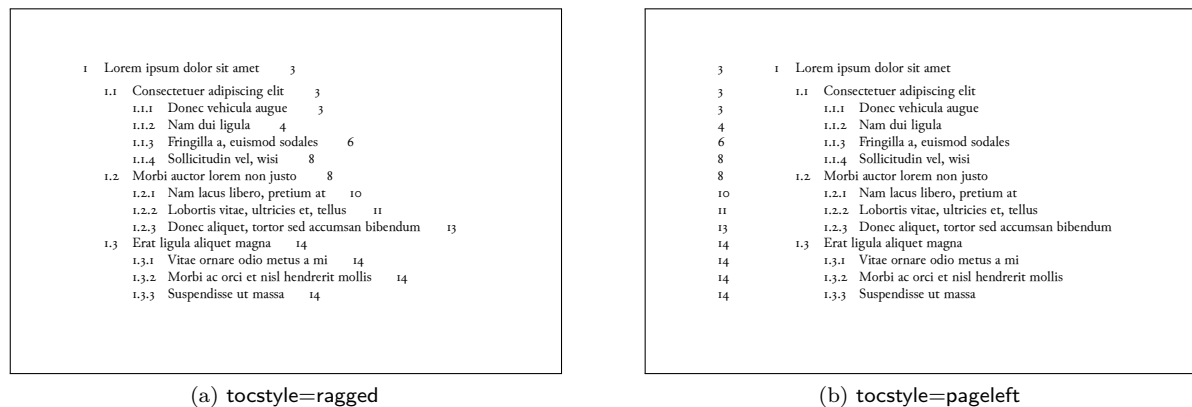


FIGURA 8: I due indici non-standard forniti dalla classe.

tocstyle= standard, dotted, ragged, leftpage

standard Produce un classico indice con i numeri di pagina incolonnati al limite destro della gabbia del testo, senza linee puntate tra voce e numero di pagina. Considerato che la gabbia di suftesi è stretta, si è deciso di impostare questo stile di default.

dotted Come il precedente, ma con linee puntate tra titolo e numero di pagina, ritenute da qualcuno importanti per facilitare la consultazione. I puntini sono omessi per parti e sezioni di primo livello.

ragged Compone l'indice allineato a sinistra, con i numeri di pagina vicini alla relativa voce, come si vede nella figura 8a.

leftpage Incolonna i numeri di pagina a sinistra, prima delle voci, come mostrato nella figura 8b.

Forma e allineamento delle voci dell'indice dipendono anche dalla larghezza dell'etichetta che contiene il numero della sezione. La classe definisce uno spazio sufficiente per ciascun livello dell'indice, considerando che le sezioni di secondo livello hanno almeno due cifre. Tuttavia, in presenza di documenti molto complessi che richiedono, per esempio, sezioni di secondo livello con etichette a tre o quattro cifre, i numeri delle sezioni potrebbero avvicinarsi troppo alla rispettiva voce fino anche

a sovrapporsi. Per questo si è pensato al seguente comando:

`\toclabelspace{<level>}{<label width>}`

Permette di impostare a `<label width>` la larghezza dell'etichetta del `<level>`, dove `<level>` è il nome del livello di sezionamento (**part**, **chapter**, **section**, ecc.).

2.2.6 Il frontespizio

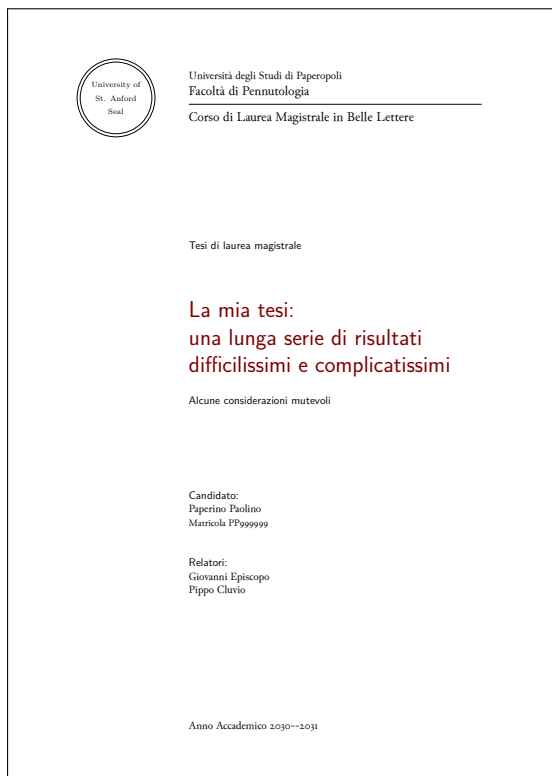
Se nelle pubblicazioni il disegno del frontespizio è tanto importante quanto il disegno degli altri elementi del progetto, nelle tesi di laurea raramente ci si discosta dallo standard del frontespizio centrato. La classe suftesi è nata con un frontespizio dedicato (figura 9), che è stato successivamente integrato nel pacchetto frontespizio (GREGORIO, 2011) ed è ora indipendente dalla classe usata.

Questo stile usa un font senza grazie per alcuni elementi e produce il titolo in color rosso scuro. Come si può vedere dalla figura, la presenza del logo è determinante per il caratteristico effetto asimmetrico. Poiché tuttavia non tutti gli atenei ne consentono l'uso, l'utente dovrà prestare particolare attenzione per evitare problemi legali.¹²

3 Stili predefiniti

Le opzioni viste finora consentono di creare diverse varianti grafiche. Considerando solo i capitoli, si hanno tre posizioni per il titolo, due posizioni per il

¹² Grazie a opportune definizioni la classe permette di usare il pacchetto frontespizio anche con i layout compatti.

FIGURA 9: Il frontespizio di *suftesi* nel layout *periodical*.

numero e tre stili di carattere. Il numero, poi, può essere composto in cifre arabe o romane, seguito o meno da un punto. Le variazioni possibili sono più di cento. Se si personalizzano poi anche le sezioni di secondo livello si arriva facilmente a migliaia di varianti.

Naturalmente, di queste varianti solo alcune sono veramente valide. Si è pensato quindi di mettere a disposizione un ampio set di stili predefiniti, che dovrebbero soddisfare le esigenze della maggior parte degli utenti.

3.1 Varianti standard

Un primo gruppo di stili predefiniti sono varianti sullo stile di default, diversi per posizione di titolo e numero di capitolo (ne abbiamo già visto qualcuno nella figura 3). Ciascuna di queste varianti può essere combinata con l'opzione `chapnumstyle` e ottenere così ulteriori effetti grafici, alcuni dei quali sono mostrati nella figura 11.

```
style= roman<1-6>, italic<1-12>,
      smallcaps<1-12>
```

roman<1-12> Titoli di capitolo e testatine sono composti in tondo. Il titolo è in linea col numero negli stili 1-3 e sotto il numero negli stili 4-6.

italic<1-12> Titoli di capitolo e testatine sono composti in corsivo. Il titolo è in linea col numero negli stili 1-3 e sotto il numero negli stili 4-6. Gli stili 7-12, che compongono le sezioni di secondo livello

in maiuscoletto, sono varianti dei primi sei.

smallcaps<1-12> Variante della precedente, che usa il maiuscoletto invece del corsivo.

3.2 Varianti elements

Una delle differenze tra lo stile *ClassicThesis* e la classe *suftesi* è che il primo si ispira agli *Elementi dello stile tipografico* (BRINGHURST, 1992), la seconda permette di riprodurre “fedelmente” il layout, sebbene questo non sia il suo scopo originario. Vedremo subito perché si sono usate le virgolette.

Com'è noto, il capolavoro del poeta canadese ha un progetto grafico alquanto ricercato: il numero di capitolo compare nel margine; al posto delle testatine ha titoli correnti; la gabbia del testo è in proporzione 1 : 2 e il layout di pagina è uno slanciato $1/\sqrt{3}$ derivato dall'esagono (figura 10). Oltre a questo è anche un capolavoro letterario, che merita d'essere letto da tutti gli amanti del buon gusto.

Il testo è composto nel font *Minion Pro*, un bellissimo carattere da testo in stile aldino disegnato da Robert Slimbach. Trattandosi di un font commerciale non presente in T_EX Live, non si può riprodurre lo stile se non acquistando i caratteri. La classe, invece, carica di default il *Palatino* e questo è il primo tradimento degli *Elementi*.¹³ Un ottimo risultato, molto vicino all'originale nell'effetto, si può ottenere con il carattere *Junicode* già ricordato.

Il numero di capitolo è composto in *Aldus*, carattere disegnato da Hermann Zapf per comporre testi e da affiancare al *Palatino* come carattere per titoli (BRINGHURST, 1992, 215, 239). Anche per questo elemento la classe carica al suo posto il più noto carattere di Zapf, che nel limite della funzione svolta, va però detto, si distingue dall'*Aldus* per dettagli trascurabili. Questo è il secondo tradimento degli *Elementi*.

Gli *Elementi*, poi, non hanno sezioni di quarto livello, ma si è ritenuto opportuno implementarle ugualmente, sebbene Bringhurst potrebbe non essere d'accordo. Ugualmente, si possono usare anche i livelli inferiori in maniera coerente (si spera) con il disegno generale.

Un problema più difficile, sebbene non a livello T_EXnico, è stato implementare l'indice generale. Nell'originale, l'indice che appare ad inizio testo (intitolato *Sommario*) riporta soltanto i capitoli, mentre le sezioni di terzo livello compaiono nell'appendice E (intitolata *Ricapitolazione*). Le sezioni di secondo livello, al contrario, non compaiono in nessun indice e ciò è perlomeno singolare. Si deve però tener presente che quelle che per noi sono

13. Una versione ridotta del *Minion Pro* in formato OpenType è contenuta in *Adobe Reader*, ma la licenza con cui viene distribuito non è chiara e quindi non è possibile consigliarlo.

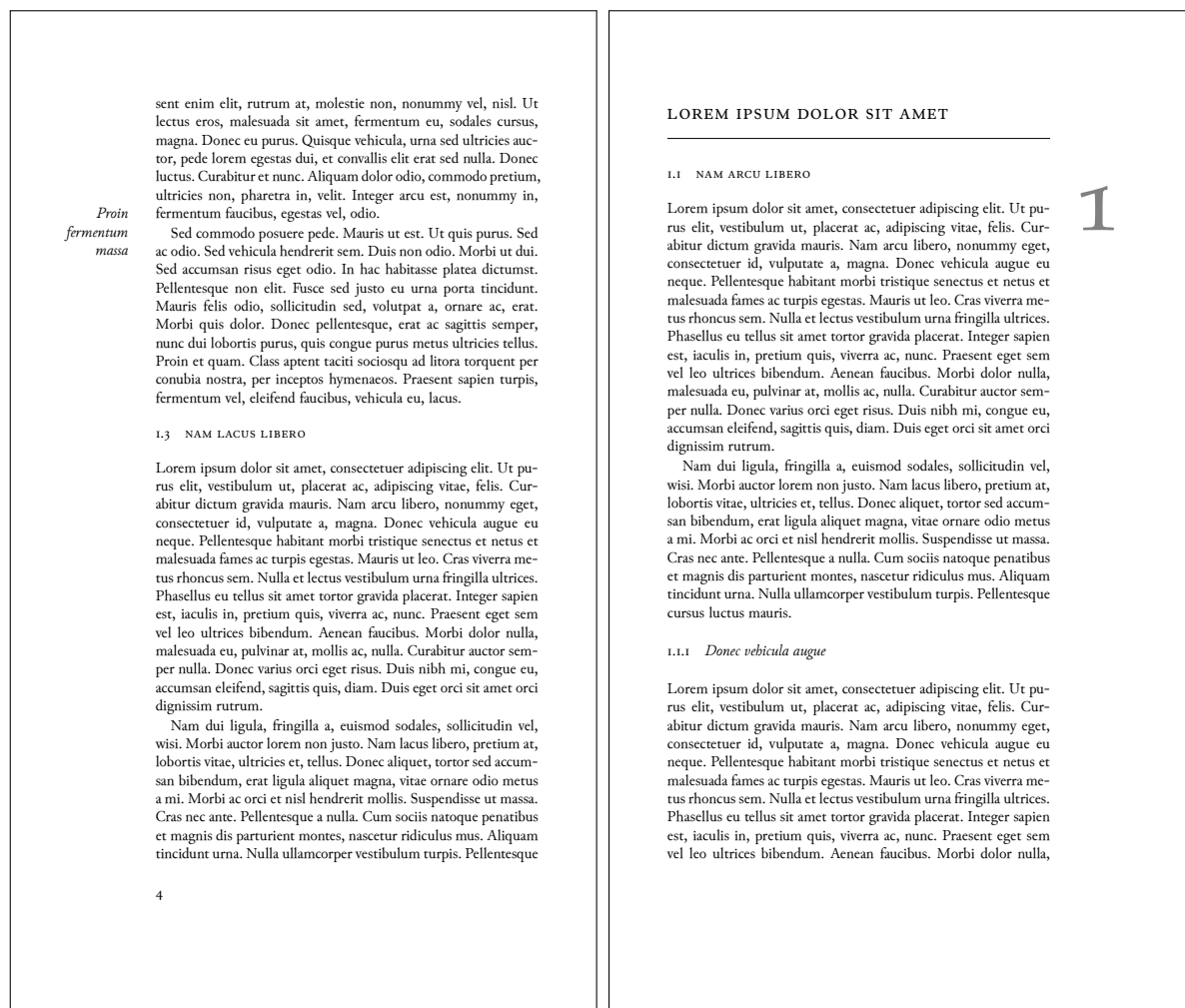


FIGURA 10: Due pagine affiancate nello stile degli *Elementi* (BRINGHURST, 1992). Quando una sezione compare in una pagina pari, come in questo caso, nel margine apparirà il titolo della sezione precedente. Un titolo che cade a inizio pagina, invece, verrà riportato anche nel margine.

sezioni di terzo livello, nel testo di Bringhurst sono (anche) delle “regole” e per questo vengono ricapitolate, come silloge, alla fine del testo. Questo è il quarto tradimento.

Per quanto riguarda le didascalie, mi sono ispirato a quelle presenti, sebbene non vi sia alcun oggetto mobile. Etichetta e testo della didascalia sono composti in corpo `\small`, in carattere *san-serif* e senza rientri particolari. Nell’originale le didascalie sono composte in Scala Sans di Martin Majoor, mentre *suftesi* usa l’Iwona. Un altro tradimento dello stile.

Di tradimenti ce ne saranno sicuramente altri, così come di omissioni e lacune: d’altra parte lo stile è stato definito a partire dalla traduzione italiana. E tradurre, come è noto, significa tradire. Credo tuttavia che non si debba pretendere tanto di più: in fondo questo stile non andrebbe mai usato, anche solo per rispetto. Per chi invece vuole essere irrispettoso, esiste l’opzione:

```
style = elements
```

L’ultimo stile predefinito è un ibrido che compone i capitoli e le sezioni in stile *elements*, ma usa

le testatine standard e può essere impiegato anche con i layout compatti:

```
style = sufelements
```

Se fosse necessario modificare il carattere del numero di capitolo, si può ricorrere al seguente comando, che richiede sintassi diverse, a seconda che si usi come motore LATEX o XYLATEX:

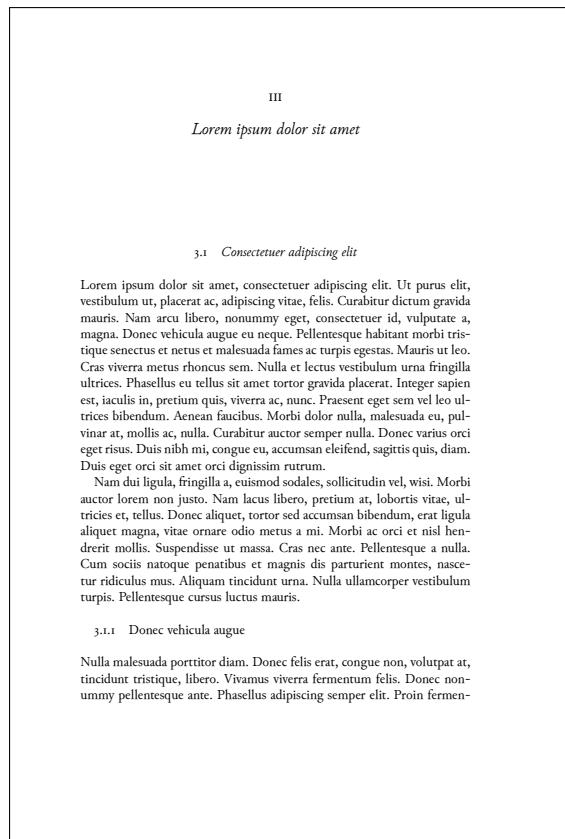
```
\chapnumfont{pplj}
```

```
\chapnumfont{%
  \fontspec[Numbers=OldStyle]{Junicode}}
```

Nel primo caso si indica il nome della famiglia, quale appare nel relativo file PostScript; nel secondo si usa invece la sintassi prevista dal pacchetto *fontspec*.

3.2.1 Titoli o spalle correnti

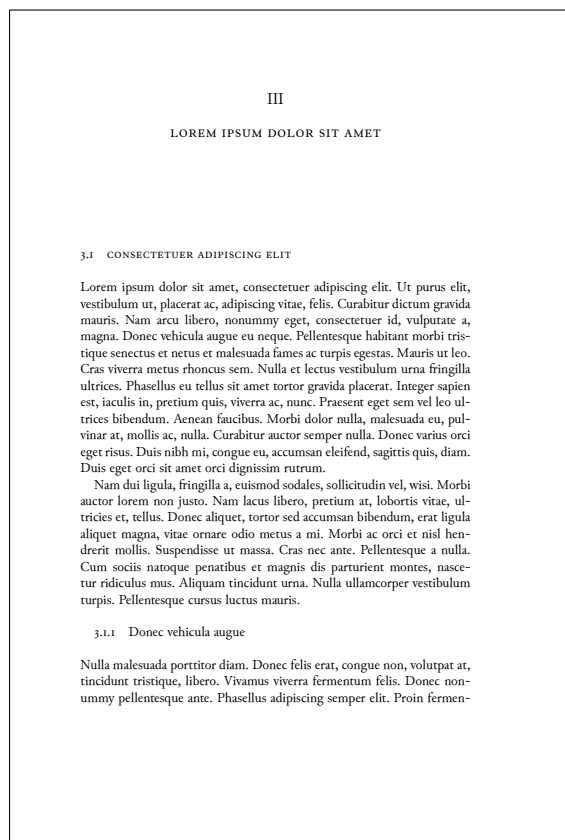
Negli *Elementi*, le informazioni generalmente sono contenute nelle testatine appaiono nella posizione delle note a margine, in quelle che vengono definite comunemente *spalle correnti*. Nelle pagine pari appare il titolo della sezione, mentre nelle pagine



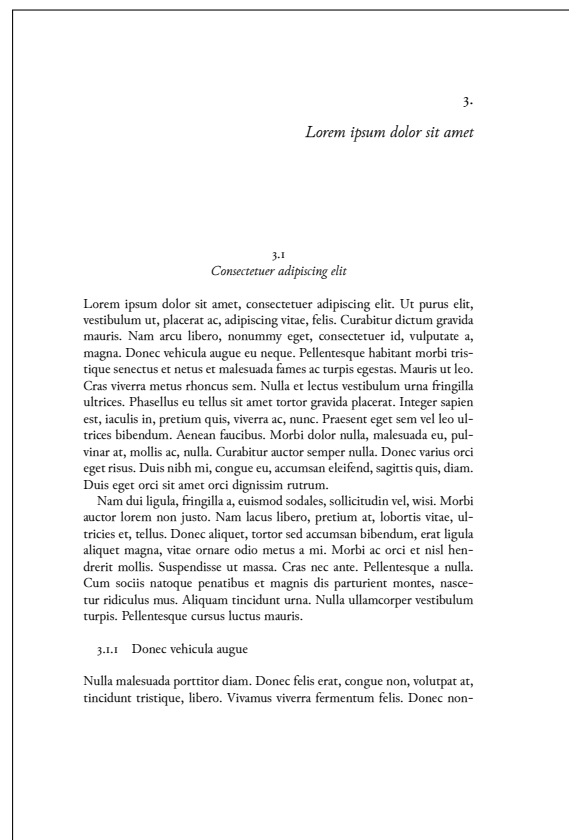
(a) style=italic5, chapnumstyle=roman, secstyle=center



(b) style=smallcaps1, chapnumstyle=dotarabic



(c) style=smallcaps11, chapnumstyle=Roman



(d) style=italic6, chapnumstyle=dotarabic, secstyle=parcenter

FIGURA 11: Varianti sugli stili italic e smallcaps. Si noti il titolo di sezione centrato in (a) e centrato sotto il numero in (d). Il numero di capitolo è composto in maiuscoletto romano in (a) e in maiuscolo romano in (c). Il punto dopo il numero è stato inserito solo in (b) e (d).

dispari il titolo del capitolo. Sono composte in corsivo e nello stesso corpo usato per il testo (figura 10).

4 *Farragines*

4.1 Colophon

Prima dell'introduzione del frontespizio, che avvenne in maniera decisa verso la fine del Quattrocento, le informazioni su autore, stampatore, tipografo, ecc. venivano inserite in una nota a fine volume che prendeva il nome di *colophon* (in greco "cima", "vertice").¹⁴ I *colophon* sono a volte "scenografici": alcuni sono in forma poetica e spesso sono composti a finalino o a sagoma.

Oggi il *colophon* appare soltanto in pubblicazioni di pregio o di editori con una certa sensibilità tipografica ed è un peccato che se ne sia persa l'abitudine. In sua assenza, il lettore che volesse conoscere il carattere usato in un testo dovrebbe rivolgersi a uno dei molti servizi online per il riconoscimento dei caratteri.¹⁵ Anche in questo caso, però, l'informazione potrebbe essere incompleta o imprecisa, perché alcuni caratteri hanno una loro storia, un nome, che prescinde dal disegno. Per esempio, si potrebbe scoprire che il font del proprio volume preferito è un Classical Garamond BT, ma solo un'ulteriore ricerca ci svelerebbe che si tratta in sostanza del carattere Sabon, disegnato da Jan Tschichold e distribuito sotto altro nome dalla Bitstream.

La classe è nata con una certa rigidità e richiede all'autore di inserire una nota di attribuzione. In altre parole impone di mettere da qualche parte un *colophon* (anche se questo andrebbe a rigore alla fine). Per questo si è pensato di fornire alcuni comandi che rendano semplice l'operazione.

```
\colophon[⟨OS⟩]{⟨nome e cognome⟩}
                {⟨informazioni aggiuntive⟩}
```

Questo comando ha tre argomenti dei quali il è primo opzionale e produce una pagina con al centro una nota sul copyright:

```
Copyright © 2012 ⟨nome e cognome⟩
Tutti i diritti riservati
```

e in fondo:

```
Questo lavoro è stato composto con LATEX su
⟨OS⟩ usando la classe suftesi di Ivan Valbusa.
⟨informazioni aggiuntive⟩.
```

Chiaramente questo comando può essere usato solo in documenti in lingua italiana. Per le altre lingue è disponibile un comando del tutto analogo al precedente, ma che richiede di inserire alcune informazioni a mano:

14. Al frontespizio e al *colophon* sono dedicate due ampie sezioni di Tuzzi (2006).

15. In questo modo si è determinata, per esempio, la vera natura del numero di capitolo degli *Elementi*.

```
\bookcolophon{⟨nome e cognome⟩}
                {⟨informazioni aggiuntive⟩}
```

Ci sono poi altri due comandi simili:

```
\artcolophon{⟨testo⟩}
\finalcolophon{⟨testo⟩}
```

Il primo stampa semplicemente il proprio contenuto a fondo pagina, mentre il secondo lo compone a epigrafe in alto ed è quindi pensato per produrre un autentico (e semplice) *colophon*.

4.2 Comandi per titoli e testatine

La classe definisce pochi nuovi comandi, che ora elencherò. Il primo è semplicemente una ridefinizione del comando standard `\title`, al quale aggiunge un argomento opzionale, utile se il titolo contiene interruzioni di riga. La sua funzione è la stessa che si ritrova nelle versioni con asterisco dei comandi di sezionamento:

```
\title[⟨titolo breve⟩]{⟨titolo lungo⟩}
```

Stampa *⟨titolo breve⟩* nelle testatine e *⟨titolo lungo⟩* nella pagina del titolo creata con `\maketitle` (che peraltro produce un titolo leggermente diverso da quello standard).¹⁶

I prossimi comandi servono invece nella revisione finale, per ottimizzare le interruzioni di riga nell'indice generale eventualmente presenti:

```
\headbreak
\xheadbreak
```

Il primo produce un'interruzione di riga attiva solo nell'indice; il secondo un'interruzione di riga attiva solo nel testo e nelle testatine. Quest'ultimo comando deve essere usato assieme all'argomento opzionale dei comandi di sezionamento, che ne elimina l'effetto nella testatina.¹⁷

Vediamo in conclusione come si usano praticamente. Immaginiamo di voler modificare una voce di capitolo che all'interno dell'indice viene spezzata in un punto non ottimale. Inserendo una semplice interruzione di riga questa avrà effetto anche nel corpo del testo e ciò potrebbe non essere quello che si desidera. Scrivendo invece:

```
\chapter{Titolo del capitolo\headbreak{}}
        che va a capo}
```

il titolo andrà a capo solo nell'indice. Viceversa, può accadere di dover spezzare un titolo solo nel testo e lasciarlo invariato nell'indice. Per questo ci viene in aiuto il secondo comando:

```
\chapter[Titolo del capitolo]
        {Titolo\headbreak{}}
        del\headbreak{}}
        capitolo}
```

16. Per produrre la pagina del titolo standard è disponibile il comando `\standardtitle`.

17. I codici per questi due comandi sono di Enrico Gregorio.

In questo modo avremo il titolo di capitolo su tre righe nel corpo del testo, ma su un'unica riga sia nell'indice sia nelle testatine.

5 Conclusione

Questa classe non ha alcuna pretesa d'essere riconosciuta in futuro come esempio di buon design. Mi auguro tuttavia che sia da stimolo alla ricerca e alla diffusione del buon gusto tipografico. «Tutto ciò che è buono per qualcosa – scriveva Vilém Flusser – è un vero e proprio male [...] il bene puro non ha senso, è assurdo, e ovunque ci sia uno scopo, c'è il diavolo in agguato» (FLUSSER, 1991, 21-22). Spero quindi di essere stato buono; ma neanche poi troppo.

Riferimenti bibliografici

- ANDREOLI, J. (2012). *L'Unione Europea in una prospettiva mereologica*. Tesi di Laurea, Università di Verona. Relatore: Ivan Valbusa.
- ARISTOTELE (s.d.). *Poetica*. Trad. it. di Domenico Pesce, Bompiani, Milano 2000.
- BRINGHURST, R. (1992). *The Elements of Typographic Style*. Hartley & Marks Publisher Inc., Vancouver, Canada. Trad. it. *Gli elementi dello stile tipografico*, Sylvestre Bonnard, Milano 2009.
- FLUSSER, V. (1990). «Vom Wort Design». *Design Report. Mitteilungen über der Stand der Dinge*, **15**. Trad. it. in *Filosofia del design*, Bruno Mondadori, Milano 2003, p. 1-6.
- (1991). «Der Krieg und der Stand der Dinge». *Design Report*, **16**. Trad. it. in *Filosofia del design*, Bruno Mondadori, Milano 2003, p. 17-22.
- GILL, E. (1931). *An Essay on Typography*. Sheed & Ward. Trad. it. *Sulla tipografia*, Sylvestre Bonnard, Milano 2005.
- GREGORIO, E. (2010). «L'arte esoterica di scrivere in cirillico con L^AT_EX». *ArsT_EXnica*, **9**, pp. 57–73.
- (2011). «Il pacchetto frontespizio». URL <http://www.ctan.org/tex-archive/macros/latex/frontespizio>. Versione 1.4a.
- LESINA, R. (2009). *Il Nuovo Manuale di Stile*. Zanichelli, Bologna.
- MADINELLI, L. (2012). *Le origini del pragmatismo nel carteggio tra Peirce e James*. Tesi di Laurea, Università di Verona. Relatore: Ivan Valbusa.
- MIEDE, A. (2012). «A Classic Thesis Style». URL <http://www.ctan.org/tex-archive/macros/latex/contrib/classicthesis>. Versione 4.1.
- MORISON, S. (1967). *First Principles of Typography*. Cambridge University Press. Trad. it. *I principi fondamentali della tipografia*, Fabrizio Serra Editore, Pisa-Roma 2008.
- MUNARI, B. (1966). *Arte come mestiere*. Laterza, Roma-Bari.
- PANTIERI, L. e GORDINI, T. (2012). *L'arte di scrivere con L^AT_EX*. URL http://www.lorenzopantieri.net/LaTeX_files/ArteLaTeX.pdf.
- SCHENA, G. (2011). *Personaggi fittizi ed entità dubbie nelle ontologie di Francesco Berto e Achille Varzi*. Tesi di Laurea, Università di Verona. Relatore: Ivan Valbusa.
- TSCHICHOLD, J. (1975). *Ausgewählte Aufsätze über Fragen der Gestalt des Buches und der Typographie*. Birkhäuser, Basel. Trad. it. *La forma del libro*, Sylvestre Bonnard, Milano 2003.
- TUZZI, H. (2006). *Libro antico libro moderno. Per una storia comparata*. Sylvestre Bonnard, Milano.
- VALBUSA, I. (2008). *La forma dell'enciclopedia. Una valutazione della prospettiva di J.H. Alsted*. Verifiche, Trento.
- (2012). «User's Guide to suftesi. A document class for typesetting theses, books and articles». URL <http://www.ctan.org/tex-archive/macros/latex/contrib/suftesi>. Versione 1.0.

▷ Ivan Valbusa
Dipartimento di Informatica
Facoltà di Lettere e Filosofia
Università di Verona
ivan dot valbusa at univr dot
it

Una classe per comporre un dizionario

Claudio Beccari, Heinrich Fleck

Sommario

In questo articolo descriviamo la storia, lo sviluppo e le principali caratteristiche delle classi `dictionarySCR` e `xdictionarySCR`, che permettono di comporre, rispettivamente con `pdfLATEX` e `XYLATEX`, dizionari tematici in una varietà di discipline.

Abstract

In this article we describe the history, the development, and the most relevant characteristics of classes `dictionarySCR` and `xdictionarySCR`, that allow to typeset, by means `pdfLATEX` and `XYLATEX` respectively, thematic dictionaries in a variety of disciplines.

1 Premessa

Nel 2010 abbiamo presentato un articolo sui *mark* (BECCARI e FLECK, 2010) (parola che non abbiamo voluto tradurre in italiano, perché ‘marca’ o ‘marchio’ non ci sembrano adeguati) che abbiamo usato in diversi modi durante la creazione della nostra classe `dizionarioSCR`, trovando poi finalmente una soluzione quasi ottimale, secondo le nostre necessità.

Nello stesso tempo il *Dizionario di Astronomia*¹, per il quale la classe veniva preparata, ha cominciato a circolare in bozza fra i cultori di questa scienza, ed è arrivato alle orecchie di un utente che aveva bisogno di scrivere un dizionario sullo scintoismo giapponese, settore disciplinare completamente diverso dall’astronomia. Per adattare la classe alle sue esigenze, cioè di inserire nei lemmi in italiano anche delle brevi locuzioni in giapponese, in cinese e in devanagari (sanscrito), abbiamo adattato la classe all’uso con `XYLATEX`. Le classi in realtà erano due distinte per lavorare l’una con `pdfLATEX` e l’altra con `XYLATEX`; il nome è cambiato in `dictionarySCR`, e finalmente è documentata (in inglese) con il file `dictionarySCR.dtx`. Per lavorare con `XYLATEX` la classe si chiama `xdictionary` ed è documentata nel file `xdictionary.dtx`. Le due classi sono quasi identiche; le differenze riguardano solo il diverso trattamento delle lingue, (con il pacchetto `babel` con `pdfLATEX` e con `polyglossia` in `XYLATEX`), e il trattamento dei font (Type 1 con `pdfLATEX` e OpenType con `XYLATEX`). Questa dupli-

1. Questo Dizionario di Astronomia non è ancora stato pubblicato, anche perché non è ancora finito; Heinrich Fleck lo ha cominciato nel 2009 e da allora vi sta lavorando sopra più per sua passione che per il desiderio di vederlo pubblicato. Chissà, forse in futuro...

cazione della classe non sarebbe necessaria se non si fossero voluti usare i font greci che con `pdfLATEX` sono gestiti attraverso il pacchetto `teubner`; questo pacchetto però, non è ancora completamente compatibile con `XYLATEX`. Per altro ora stiamo lavorando ad una unificazione delle due classi in una sola, in modo che sappia gestirsi autonomamente con entrambi i programmi di composizione.

Vale la pena di notare che le classi “inglesi” sono nate dalla originaria classe “italiana” e questo si riflette nel fatto che tutti i comandi specifici delle classi `dictionarySCR` e `xdictionarySCR` sono bilingui, come per esempio `\vedilemma` e `\seeentry`.

Qui raccontiamo le nostre esperienze e mostriamo alcune parti del codice. Lo scopo di questo articolo è mostrare che cosa la classe sia capace di fare. La classe non è ancora resa pubblica, perché secondo noi è ancora al livello “alpha”, non ancora al livello “beta”. Può darsi che la classe e un “campione” del Dizionario compilato ma fornito assieme al codice sorgente, vengano posti a disposizione dei soci del `GYLATEX`, nella sezione dei “Template”, di cui si è parlato nei giorni in cui stavamo scrivendo questo articolo.

2 Specifiche richieste per la compilazione dizionario

Il nostro dizionario doveva essere composto su carta in formato A4, con una *front matter* composta a piena pagina, una *main matter* composta a due colonne, e una *back matter* di nuovo composta a piena pagina, in quanto avrebbe contenuto delle grandi e lunghe tabelle contenenti degli estratti dai cataloghi di stelle. La bibliografia e l’indice dei nomi degli astronomi e di vari altri personaggi storici, che avevano avuto importanza diretta o indiretta in astronomia, andavano messi nella parte finale del dizionario come ultimi elementi, entrambi composti di nuovo a due colonne.

I lemmi del dizionario, ordinati alfabeticamente, sarebbero stati contenuti in apposite sezioni, al livello dei capitoli, il cui titolo consistesse nell’iniziale corrispondente. Tale lettera sarebbe dovuta apparire anche in ogni pagina del “capitolo” come `/emphunghia`² posta nel margine esterno sia sulle pagine di destra sia su quelle di sinistra.

2. In certi dizionari, in molte Bibbie, spessissimo in quel tipo di libretto che chiamiamo “rubrica”, dove il materiale è separato secondo la lettera iniziale oppure secondo i nomi abbreviati convenzionali dei Libri Sacri, queste informazioni vengono stampate nel margine esterno delle pagine di destra; spesso le pagine precedenti hanno un intaglio semicircolare in modo che il simbolo della lettera o del Libro Sacro appaia

Siccome il dizionario avrebbe finito per essere consultato prevalentemente come documento PDF direttamente sullo schermo, abbiamo ritenuto opportuno realizzare i riferimenti incrociati dei lemmi mediante collegamenti ipertestuali. Usare il pacchetto `hyperref` implica assoggettarsi alle sue idiosincrasie, perfettamente giustificate quando se ne conosca il meccanismo interno, ma talvolta un poco fastidiose; fra queste idiosincrasie c'è quella che tutto ciò che deve venire collegato in modo ipertestuale deve essere identificato in modo univoco, ovviamente, e `hyperref` risolve questo problema usando i numeri degli oggetti che il compilatore ha assegnato loro anche se questo numero non compare nel documento finale; per esempio: la terza equazione (numerata) che compare nel trecentoquindicesimo lemma (il cui numero non appare) della sezione 'C' riceve dal programma di compilazione l'identificativo univoco 'C.315.3' e questa stringa, oltre ad altre informazioni, viene usata da `hyperref` per costruire l'ancora e il bersaglio (*anchor* e *target*) dei riferimenti ipertestuali. I comandi `\label`, `\ref` e `\pageref` (oltre a quelli definiti dal pacchetto `varioref` il cui funzionamento, però, non è affidabile come quello dei comandi originali di LATEX) vengono ridefiniti da `hyperref` in modo da contenere e interpretare anche le informazioni accessorie necessarie per la costruzione corretta delle ancore e dei bersagli.

Le figure e certe tabelle possono avere delle didascalie senza un numero visibile, ma debbono avere un'ancora univoca legata al numero invisibile del lemma a cui si riferiscono. Le didascalie delle figure sono collocate sopra o sotto la figura a seconda della scelta del compositore; l'idea è che se una figura o una tabella compare in testa ad una colonna, deve o dovrebbe avere la didascalia di sopra, mentre se cade alla base della colonna la didascalia dovrebbe apparire sotto l'oggetto.

Abbiamo previsto anche che alcune tabelle, più larghe di una colonna, ma non tanto larghe da richiedere uno spazio a piena pagina, possano estendersi nel margine esterno; perciò abbiamo dovuto ideare un meccanismo per scoprire se la figura si trova nel punto giusto e si estenda nel margine giusto. Visto che il testo è a due colonne, nelle pagine di sinistra la tabella in questione può estendersi solo a sinistra e quindi essere nella colonna di sinistra, mentre nelle pagine di destra deve succedere il contrario.

Le figure delle pagine 33 e 36 mostrano due esempi di pagine affiancate del dizionario in cui vengono

anche a libro chiuso. Evidentemente questo tipo di informazioni serve per trovare l'inizio della sezione che interessa, senza bisogno di consultare un indice; la forma dell'intaglio ricorda quello di un'unghia e per traslato si chiama così non solo l'intaglio, ma anche il segno che l'intaglio lascia apparire; il segno conserva il nome di unghia anche se l'intaglio manca. In inglese la parola *thumbnail* indica lo stesso concetto, e addirittura fa riferimento all'unghia del pollice.

mostrate molte delle caratteristiche descritte in questo paragrafo.

In particolare si osserva la posizione dell'unghia, una figura a piena pagina e due figure in colonna con la didascalia sopra o sotto a seconda della scelta del compositore; si vede un medaglione a due colonne una delle quali è scritta in greco classico, mentre l'altra contiene la traduzione. Le testatine contengono i nomi dei lemmi in vigore all'inizio della colonna di sinistra e alla fine della colonna di destra, con il numero della pagina al centro. Questa particolare coppia di pagine non contiene equazioni, ma contiene una piccola tabella a sfondo colorato alla fine della seconda colonna della seconda pagina; contiene riferimenti ipertestuali ad altri lemmi che, nella versione originale, hanno i testi cliccabili per spostare la visione sul lemma citato e sono colorati di rosso; nella stampa in bianco e nero appaiono in grigio chiaro.

Invece nello *spread* della pagina 36 si possono osservare sia una tabella sporgente e con lo sfondo colorato, sia, nella voce "atlante", dei titolini marcati mediante un triangolo nero con la punta a destra; all'inizio della voce "atlante" c'è anche il mini indice relativo a questo lemma, ma siccome la sua descrizione occupa diverse pagine affiancate, non lo si è riportato completamente per ovvi motivi di economia di spazio.

3 Soluzioni trovate

Dopo diverse prove abbiamo trovato che la classe più adatta su cui costruire la classe `dictionary-SCR` era la classe `scrbook` della collezione Koma Script. Abbiamo provato anche la classe `standardbook`, che avrebbe però richiesto un lavoro maggiore, e la classe `memoir` (WILSON, 2011), altamente configurabile, ma con le sue rigidità.³

La collezione delle classi Koma Script (KOHM e MORAWSKI, 2012) dispone di un comodo comando `\areaset` con il quale è possibile stabilire o modificare la geometria dello specchio di stampa con facilità. Quindi non è stato difficile creare le geometrie per le pagine iniziali, intermedie e finali, che si è rivelata un'operazione di routine.

3.1 I titoli dei capitoli, i mark e le unghie

Più delicata è stata l'operazione di comporre l'unghia con la lettera iniziale del gruppo di lemmi; questo ha richiesto la ridefinizione del comando `\@chapter` con il quale viene effettivamente composto il titolo del capitolo, affinché non scriva il nu-

3. Tra le altre cose volevamo usare la procedura fornita dal pacchetto `imakeidx` (BECCARI e GREGORIO, 2012) per comporre l'indice dei nomi sempre in modo sincrono con la compilazione del dizionario, e a quel tempo la classe `memoir` e il pacchetto `imakeidx` erano ancora incompatibili; ora `imakeidx` contiene il necessario per lavorare anche con `memoir`, ma ormai siamo troppo avanti per ricominciare daccapo ed adattare la nostra classe all'uso di `memoir`.

Palinsesto di Archimede: a destra il palinsesto, a sinistra la parte di testo evidenziata nel rettangolo rosso sovrascritta dalle preghiere; Walters Art Museum di Baltimora

[illegible]

...e un metodo ideato per la misura del diametro apparente del
 sole,⁴ e forse anche questa tecnica conflui in un'opera. L'opera

Osservando l'astro al sorgere, Archimede pose su un'asta un cilindro che poteva scorrere lungo questa, in modo che posto il cilindro fra l'occhio e il Sole fosse possibile (avvicinandolo e allontanandolo) o vedere solo una debole luce ai lati del solido o nascondere completamente il sole. Misurati i due angoli sottesi dalle diverse posizioni del cilindro con

che nell'antichità dovette godere di notevole popolarità tanto che se ne ha un'eco in Cauallo ed Orazio (anche se quest'ultimo la accredita ad Archita), è l'unica d'Archimede che tratti questioni astronomiche.

Muovendo dalla concezione avanzata da **Aristarco** in cui questi, in un'opera andata perduta, presentava come *ipotesi* un modello geocentrico, argomentando dalle teorie dell'astronomo di Samo, Archimede comprende come un simile modello comporti una ridefinizione della struttura e delle dimensioni dell'universo (la sfera) come era allora conosciuto, e per conseguenza la necessità di esprimersi in grandezze del tutto fuori dal comune.

L'astronomia si è infatti solo uno spunto per svolgere il discorso sui grandi numeri. Sotto questo aspetto l'*Avenario* assomiglia un poco al *Problema dei buoi* che non ha affatto una finalità ludica, ma fondamentalmente matematica, trattandosi anche in questo caso di grandi numeri.

non è possibile ritenere che esso abbia alcun rapporto rispetto alla superficie della sfera), e il suo è più che altro un discorso di formalismo matematico.

«L'esigenza è in via prioritaria quella di concepire ed esprimere grandi numeri utilizzando l'antico sistema di numerazione greca che scriveva i numeri con lettere e non conosceva lo 0, circostanza che rendeva la lettura delle opere di Archimede come di altri autori del periodo greco ed ellenistico particolarmente ardua, giacché non solo i numeri, ma anche le frazioni e le equazioni sono espresse in modo del tutto diverso da come siamo abituati oggi a scriverle ed usarle: ➔ **astronomia greca**.

La precisione costituisce la chiave di volta di lettura dell'opera, perché ad ammettere la scrittura secondo il nostro sistema numerico-decimale, il problema perderebbe gran parte della sua rilevanza. Questa difficoltà si rinviene nelle prime righe dell'*Arenario* (vedi riquadro a fronte), ed il ricordato libro intitolato a Zeupisio poteva anzi essere trattato forse anche dei calcoli oltre che dei grandi numeri.

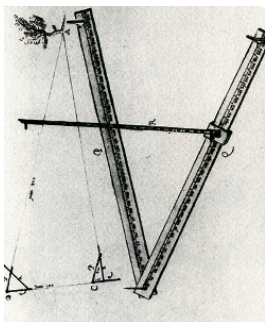
fine di Archimede è rappresentare un numero che (per quanto è possibile) si avvicini a un numero intero, e che non sia esprimibile (per quanto è possibile) come rapporto di due numeri naturali. Archimede si affrettava a dire che i numeri naturali non sono infiniti, e che non si può contare fino all'infinito. Ma, per quanto riguarda la possibilità di intelligenza della mente umana (... con dimostrazioni geometriche che potrai seguire con facilità, e con un po' di *pena*...), dominare il mondo fisico tramite la matematica, e dimostrare che è possibile immaginare e scrivere un numero più grande del numero dei granelli d'arena che potrebbero essere contenuti nell'universo: Archimede non confuta né abbraccia le proposizioni di Aristarco, considera soltanto le dimensioni dell'universo adottando nuovi simboli per numeri più grandi.

Dopo aver descritto il metodo di numerazione, Archimede dimostra un teorema sulle proporzioni (che grosso modo esprime la legge di equivalenza $10^m \times 10^n = 10^{m+n}$ che a molti ha fatto credere che si fosse ad un passo dall'ideazione dei logaritmi), e passa quindi al problema dei granelli di sabbia vero e proprio.

La prima problema dei greci fu ancora vero e proprio. Non riportò qui tutti i passaggi che svolge Archimede che richiederebbero una trattazione a parte, ricordo soltanto che la miriade di miriadi (10^8) è considerata l'unità del suo sistema di numerazione, e che chiama *numeri primi* (ma con significato diverso dal nostro) quelli che vanno da 1 a 10^8 , *numeri secondi* quelli che vanno da 10^8 a $10^8 \times 10^8$ (10^{16}), *numeri terzi* quelli che vanno da 10^{16} a $10^8 \times 10^8 \times 10^8$ (10^{24}), e prosegue così con i numeri quarti, numeri quinti... fino a che l'ordine non diventa una miriade di miriadi.

Archimede poi costruisce una sfera immaginaria con il diametro uguale alla presunta distanza terra-Sole ed ipotizza una proporzione fra diametro Terra/diametro Sfera e diametro Sfera/diametro Universo (la sfera delle stelle fisse), calcolando il

Archimetro

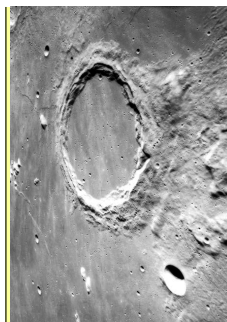


di diametro dell'universo ed il volume, rispettivamente, in 10^{14} ed in 10^{26} , per cui la quantità di granelli sarebbe $10^{12} \times 10^{21}$ e cioè 10^{33} . Archimede non si ferma qui. Costruisce numeri sempre più grandi che poi riduce ad unità di ordini superiori e poi gli ordini si riunisce in periodi sino a raggiungere alla miriade di miriadi di miriadi del miriadesimo ordine della miriade del miriadesimo periodo e cioè $10^{10.000.000.000.000.000}$ e cioè $100.000.000 \times 1$ seguito da 800.000.000 zeri.

Archimedes, cratere Cratere lunare di notevole estensione (83 km) situato al limite orientale del Mare Imbrium. A sud del cratere si estende la catena dei Montes Archimedes per circa 150 km, è privo del picco centrale.

Archimetro Strumento inventato e descritto da G. B. Alcoli, architetto fra i più rilevanti a cavallo fra il Cinquecento e il Seicento italiano. Era composto da una tavoletta triangolare con varie scale di misura. Montato su un supporto e fornito di una **diottra**, permetteva di tracciare oggetti lontani.

Archipendolo Strumento per controllare la perfetta orizzontalità di un piano. Era composto di due regoli incernierati dal cui centro pendeva il filo a piombo. Un arco graduato con lo 0 al centro indicava i gradi eventuali di disallineamento.



λ	φ	mld anni	d	h	h/d
4°W	27°7' N	3,8 - 3,2	83 km	2,15 km	0,0259

Dati del cratere Archimedes

mero ma nello stesso tempo definisca il significato del comando `\unghia`:

```

1 \newcommand*\unghia{}
2
3 \def\chapter[#1]#2{%
4   \ifnum \c@secnumdepth >\m@ne
5     \if@mainmatter
6       \refstepcounter{chapter}%
7       \typeout{\@chapapp\space
8         \thechapter.}%
9       \addcontentsline{toc}{chapter}{#1}%
10      \fi
11    \fi
12    \chaptermark{}%
13    \if@twocolumn
14      \if@at@twocolumn
15        \@makechapterhead{#2}%
16      \else
17        \@topnewpage[%
18          \@makechapterhead{#2}]%
19      \fi
20    \else
21      \@makechapterhead{#2}%
22      \@afterheading
23    \fi
24    \label{chap:#1}%
25    \def\unghia{#1}%
26 }
```

Nel codice precedente la maggior parte delle righe sono repliche inalterate del comando interno della classe `scrbook`; le poche, ma essenziali modifiche riguardano la riga 12, dove il comando `\chaptermark`, che serve normalmente per scrivere nella testatina di sinistra il titolo del capitolo, non produce nessun testo per la testatina a cui penseranno i comandi per inserire i lemmi. Invece le righe 24 e 25 sono state aggiunte; il comando `\label` serve per attribuire una etichetta di default ad ogni capitolo; l'etichetta è formata dalla stringa `chap:` agglutinata alla lettera iniziale (maiuscola) del lemmi di quel capitolo. Il successivo comando definisce il contenuto dell'*unghia*, di quella lettera nel margine esterno che aiuta a vedere subito in quale sezione di lemmi ci si trovi⁴. Tutti gli altri comandi un po' criptici non meritano una spiegazione che, per altro, si trova nei file `.dtx` di quasi tutte le classi, nel nostro caso della classe `scrbook`.

Questo, però, non basta; bisogna anche definire le testatine affinché usino correttamente l'informazione contenuta in `\unghia`. A questo scopo abbiamo ridefinito il comando che definisce lo stile `headings`, cambiandogli anche il nome, per usare la testatina come punto di riferimento fisso a cui agganciare la posizione dell'*unghia* da collocare per mezzo dei comandi dell'ambiente `picture`; si è usato un ambiente `picture` di dimensioni nulle e poi

4. Si veda la nota 2.

si è usato `\put` per collocare l'*unghia* nel punto preciso della pagina in cui si desidera che appaia.

Naturalmente questa ridefinizione delle testatine non serve solo per collocare l'*unghia*, ma anche per mettere i mark giusti nella testatina; questa deve contenere un mark a sinistra, il numero della pagina al centro e l'altro mark a destra: a sinistra il titolo della prima voce della colonna di sinistra o della voce che è ancora in vigore all'inizio di quella colonna, mentre a destra il titolo della voce la cui descrizione è in vigore alla fine della colonna di destra⁵. È noto, però, che nella composizione a due colonne, la routine di uscita non gestisce bene i mark e, per ovviare, è necessario apportare una modifica ai comandi interni di L^AT_EX, fornita dal pacchetto `fixltx2e`. Nel caso del dizionario non basta; mentre il `\leftmark` è risultato sempre corretto, il `\rightmark` non lo è stato sempre; per correggere questo difetto abbiamo dovuto definire il comando `\getcorrectrightmark` come appare nel codice che segue. Per questo la definizione dei comandi `\getcorrectrightmark` e `\ps@dizionario`⁶, sono stati definiti nel modo seguente:

```

1 \newcommand*\getcorrectrightmark{%
2   \let\protect\noexpand
3   \edef\@tempA
4   {\expandafter\@leftmark\topmark
5     \@empty\@empty}%
6   \edef\@tempB
7   {\expandafter\@leftmark\botmark
8     \@empty\@empty}%
9   \let\protect\relax
10  \ifx\@tempA\@tempB\leftmark
11  \else\rightmark\fi}
12 %
13 % nuovo stile di pagina «dizionario»
14 %
15 \def\ps@dizionario{%
16   \let\mkboth\gobbletwo
17   \let\chaptermark\gobble
18   \let\sectionmark\gobble
19   \def\@oddhead{%
20     \underline{\makebox[\textwidth]{\strut
21       \bfseries\makebox[\z@][l]{%
22         \getcorrectrightmark}\hfill
23         \thepage\hfill\makebox[\z@][r]{%
24           \leftmark}%
25         \begin{picture}(0,0)%
26           \put(20,-2){\makebox(0,0)[lb]{%
27             \HUGE\sffamily\mdseries\unghia}}%
28           \end{picture}}}%
29   \def\@evenhead{%
```

5. Nel manuale della classe `memoir`, così come nelle documentazioni di altri pacchetti e classi, per il mark di sinistra viene normalmente indicato di scrivere il nome del primo lemma che cade nella pagina; secondo noi questo non è corretto, specialmente in presenza di lemmi lunghi.

6. Questo è lo stile di pagina, simile allo stile `headings`, adattato alle necessità del dizionario; ne abbiamo creato anche l'alias in inglese, `dictionary`.

```

30 \underline{\makebox[\textwidth]{%
31 \begin{picture}(0,0)%
32 \put(-20,-2){\makebox(0,0)[rb]{%
33 \HUGE\sffamily\mdseries\unghia}}}%
34 \end{picture}%
35 \strut\bfseries\makebox[\z@][l]{%
36 \getcorrectrightmark}\hfill\thepage
37 \hfill\makebox[\z@][r]{\leftmark}}}%
38 \def\@oddfoot{}%
39 \def\@evenfoot{}%
40 }
41 % Lo stile della pagina può essere
42 % chiamato indifferentemente dictionary
43 % o dizionario
44 \let\ps@dictionary\ps@dizionario

```

La definizione di `\getcorrectrightmark` recupera i contenuti di sinistra dei mark chiamati `\topmark` e `\botmark` e li confronta; lavorare con i mark è complesso, ma la spiegazione completa appare nel nostro precedente articolo (BECCARI e FLECK, 2010) e vi si rinvia il lettore. Lo scopo di questo confronto è appunto quello di usare per il mark di sinistra il contenuto della voce in vigore all'inizio della colonna di sinistra di ciascuna pagina, e non quello della prima voce che compare nella colonna di sinistra.

Nella definizione dello stile di pagina (il cui nome inglese *page style* giustifica il prefisso `ps@` nel nome della macro che definisce ogni stile di pagina) si noti che in entrambe le testatine (`\@oddhead` la testatina delle pagine dispari e `\@evenhead` per le pagine pari) i mark sono collocati dentro scatole di larghezza nulla (righe 21, 23 e 35, 37 del codice), affinché il numero della pagina resti sempre al centro della testatina e non si sposti a destra o a sinistra a seconda della lunghezza del testo dei mark. Probabilmente questo stesso risultato si sarebbe ottenuto usando il pacchetto `fancyhdr` (VAN OOSTRUM, 2004), ma avevamo già risolto il problema da soli.

Nella parte di sinistra delle testatine delle pagine pari e, viceversa, nella parte di destra delle testatine delle pagine dispari, compare l'apertura e la chiusura dell'ambiente `picture` che definisce un disegno di dimensioni nulle; è vero che questo ambiente è poco alla moda, ma è l'ambiente grafico nativo di L^AT_EX che permette di fare in modo semplicissimo operazioni utili, come quella qui eseguita di mettere l'unghia fuori della testatina vera e propria, nel margine esterno, in una posizione fissa di 20 punti verso l'esterno e ribassata di 2 punti rispetto alla linea di base della testatina; in sostanza la lettera che costituisce l'unghia viene collocata dove si vuole senza che il disegno che la "contiene" che è di dimensioni nulle, occupi spazio nella testatina stessa.

Di conseguenza mentre lo stile delle pagine per la parte iniziale rimane quello definito dalla classe di supporto `scrbook`, per la parte intermedia e per

la parte finale bisogna provvedere ridefinendo i comandi `\mainmatter` e `\backmatter`:

```

1 \renewcommand*\mainmatter{%
2 \if@twoside\cleardoubleoddpag
3 \else\clearpage\fi
4 \@mainmattertrue\pagenumbering{arabic}%
5 \pagestyle{dictionary}%
6 \renewcommand*\chapterpagestyle
7 {dictionary}%
8 \twocolumn
9 }
10 \renewcommand*\backmatter{%
11 \if@openright\cleardoubleoddpag
12 \else\clearpage\fi\@mainmatterfalse
13 \pagestyle{headings}
14 \renewcommand*\chapterpagestyle{plain}
15 }

```

3.2 I lemmi

I lemmi sono trattati come paragrafi da introdurre con l'equivalente di un comando `\section` e sono quindi numerati, anche se il numero corrispondente non sarà visibile.

Particolare attenzione, inoltre, richiedono quei lemmi che, come spesso accade, contengono termini con diacritici, o traslitterati nell'alfabeto latino da altri alfabeti, o scritti direttamente con un alfabeto diverso da quello latino. Infatti, il pacchetto `hyperref` non permette di gestire agevolmente i caratteri speciali richiesti e quindi il compositore (l'utente) deve provvedere un titolo alternativo privo di diacritici e scritto con l'alfabeto latino. In altre parole il titolo alternativo deve essere scritto con i caratteri ASCII a 7 bit, indipendenti da qualsiasi codifica di ingresso: tale titolo alternativo viene specificato nell'argomento opzionale del comando `\lemma`; per esempio il lemma relativo a Ångström Anders Jonas, va introdotto con:

```

\lemma[Angstrom Anders Jonas]%
{Ångström Anders Jonas}

```

Inoltre per fare riferimento a un lemma, visto che non si può fare riferimento ad un numero, bisogna definire un comando `\vedilemma` che eventualmente faccia riferimento al nome senza diacritici, e in ogni caso produca un testo cliccabile che contenga il nome con diacritici. Perciò per i comandi `\lemma` e `\vedilemma` sono necessarie diverse impostazioni e ridefinizioni dei comandi interni di `scrbook`:

```

1 \newlength\abovesectionskip
2 \abovesectionskip=3.5ex \@plus 1ex
3 \@minus .2ex
4 \newlength\belowsectionskip
5 \belowsectionskip=2.3ex \@plus .2ex
6
7 \renewcommand\section{%
8 \@startsection{section}{1}{\z@}%
9 {-\abovesectionskip}%

```



```

10 {\belowsectionsip}%
11 {\ifnum \scr@compatibility>
12   \nameuse{scr@v02.96}\relax
13   \setlength{\parfillskip}
14     {\z@ plus 1fil}\fi
15   \raggedsection\normalfont\sectfont
16   \nobreak\size@section}%
17 }
18
19 \newcommand\lemma[2] [] {%
20 \ifx\currentlemma\empty
21   \def\currentlemma{#2}\fi
22 \markboth{\currentlemma}{\currentlemma}%
23 \ifblank{#1}{%
24   \label{lm@#2}\section{#2}}{%
25   \label{lm@#1}%
26   \section{\texorpdfstring{#2}{#1}}}%
27 \markboth{#2}{#2}%
28 \def\currentlemma{#2}\ignorespaces}
29 \let\paragraph\lemma\let\entry\lemma
30
31 \newcommand*\vedilemma[2] [] {%
32 \ifblank{#1}{\{\hyperref{lm@#2}{#2}\}}{%
33   {\hyperref{lm@#1}{#2}}}%
34 \let\seeentry\vedilemma

```

I riferimenti ai numeri delle equazioni, che ricominciano da 1 ad ogni lemma, devono avere un testo cliccabile che faccia riferimento anche al titolo del lemma. Bisogna sfortire il riferimento interno generato dal comando `\label` che è stato pesantemente modificato dal pacchetto `hyperref`; questo riferimento interno contiene 5 campi o argomenti racchiusi fra graffe e bisogna essere in grado di accedere agli argomenti giusti; ecco quindi che bisogna definire una macro, qui chiamata `\lemmaref` che contenga solo al terzo argomento del comando interno. Il compositore dovrebbe usare semplicemente `\equref` ed otterrebbe in lingua l'equivalente di “equazione 1 sub lemma **Alpha Carinae**”, con le parti in neretto cliccabili. Qui l'intera definizione di `\equref` svolge questi compiti: nella riga 1, tramite il comando speciale `\ifcsdef`, disponibile mediante il pacchetto `etoolbox` (LEHMAN, 2011), si verifica se al momento dell'esecuzione del comando interno l'etichetta a cui fare riferimento è associata al riferimento interno con i cinque argomenti. Il risultato di questo test decide se eseguire i comandi del primo gruppo (righe da 2 a 8) o quello del secondo (riga 9). Se sono da eseguire i comandi del primo gruppo definisce la macro `\lemmaref` con il contenuto del terzo argomento (vedi più sotto nelle righe 14 e 15, la definizione di `\LemmaRef`) che contiene il titolo del lemma. Usa poi questa nuova macro per definire l'argomento da passare al comando `\hyperref`, il comando definito dal pacchetto `hyperref` che crea il collegamento ipertestuale. Si usano le definizioni ‘espanse’ per usare i contenuti correnti delle varie macro; questa è una delle forme di programmazione accessibile solo me-

dante i comandi primitivi di TEX e non si possono usare i comandi di LATEX.

```

1 \newcommand*\equref[1]{\ifcsdef{r@#1}{%
2 \edef\lemmaref{\expandafter\LemmaEqu
3   \csname r@#1\endcsname}%
4 \edef\argomentoperhyperref{%
5   [lm@#1\lemmaref]{\lemmaref}}%
6 \textbf{\autoref{#1}} \ofentry\
7   \textbf{\expandafter\hyperref
8     \argomentoperhyperref}%
9 }{eq.~??}}
10
11 \def\diz@thirdoffive#1#2#3#4#5{#3}
12 \def\diz@secondoffive#1#2#3#4#5{#2}
13 \def\diz@firstoffive#1#2#3#4#5{#1}
14 \def\LemmaEqu#1{%
15   \expandafter\diz@thirdoffive#1

```

Complicato? Sì, ma funziona correttamente.

3.3 Tabelle che fuoriescono nel margine esterno

Sono stati creati sia un comando sia un ambiente per collocare le tabelle leggermente più larghe di una colonna in modo che fuoriescano nel margine esterno. Ciascuno presenta vantaggi e svantaggi; l'ambiente consente di mantenere la tabella mobile inserendola dentro un ambiente `table` e, almeno al secondo passaggio del compilatore, dovrebbe lasciar sporgere la tabella dal lato giusto rispetto alla parità del numero della pagina, non necessariamente rispetto alla colonna esterna che non si può conoscere prima dell'esecuzione della routine d'uscita. Il comando, invece, non produce una tabella mobile, e l'utente deve scegliere la posizione nel file sorgente in modo che la tabella sporga dal lato giusto.

Per risolvere il problema avremmo potuto ricorrere al pacchetto `wrapfig`, ma quando abbiamo scritto questa classe questo pacchetto non era ancora documentato convenientemente. Oggi potremmo riscrivere questa parte della classe, ma siccome quel che abbiamo fatto funziona in modo soddisfacente, riteniamo che non sia il caso di modificare ciò che funziona abbastanza bene. Ricordiamo inoltre che le figure avvolte dal testo di `wrapfig` presentano non pochi inconvenienti descritti dall'autore del pacchetto nella sua documentazione. Non dico che la nostra soluzione sia migliore di quella di Donald Arsenau (ARSENAU, 2003), ma presenta difetti diversi e, forse perché ci abbiamo fatto l'abitudine, ci sembrano meno importanti di quelli di `wrapfig`.

Qui ci limitiamo a descrivere l'ambiente, che funziona pressappoco come `tabular` e deve essere inserito dentro un ambiente `table` con il quale si può inserire la didascalia al solito modo. L'uso di questo ambiente segue la sintassi seguente:

```

\begin{table}
\caption[(didascalia breve)]{(didascalia)}

```

```

\label{\etichetta}
\begin{Wtabular}{\specifichedellecolonne}
\filetti e righe della tabella
\end{Wtabular}
\end{table}

```

Al posto della didascalia numerata, che si ottiene con il comando `\caption`, si possono usare, senza argomento facoltativo, i comandi `\scaptiona` e `\scaptionb` che consentono di comporre una didascalia senza numero e collocata sopra (*above*) o sotto (*below*) l'oggetto a cui si riferisce.

Nel codice che realizza questo ambiente prima bisogna aver definito alcuni registri dimensionali e un comando il cui contenuto è il numero della pagina corrente, se è la prima volta che l'ambiente viene usato; altrimenti è il secondo argomento della lista interna di argomenti generati dal comando `\label` con la sua ridefinizione da parte di `hyperref`. Dopo, l'ambiente `Wtabular` è definito in modo da farne l'uso corretto così da far sporgere la tabella dal lato giusto:

```

1 \newsavebox{\widebox}
2 \newcount\t@bella \t@bella=\z@
3 \newcount\p@gina
4 \def\PaginaTabella#1{\ifx#1\relax
5   \value{page}\else
6   \expandafter\diz@secondoffive#1\fi}
7
8 \newenvironment{Wtabular}[2][c]{%
9   \widet@ble\linewidth
10  \advance\widet@ble\columnsep
11  \advance\widet@ble\marginparwidth
12  \global\advance\t@bella\@ne
13  \edef\@tempB{dzt@the\t@bella}%
14  \expandafter\label
15  \expandafter{\@tempB}%
16  \begin{lrbox}{\widebox}%
17  \begin{tabular}{#2}%
18  }{%
19  \end{tabular}\end{lrbox}%
20  \p@gina=
21  \expandafter\PaginaTabella%
22  \expandafter{
23  \csname r@\@tempB\endcsname}%
24  \ifodd\p@gina
25  \def\@tempA{1}
26  \else
27  \def\@tempA{r}
28  \fi
29  \ifdim\wd\widebox>\widet@ble
30  \ClassWarning{dictionarySCR}%
31    {Table is too wide!}%
32  \fi
33  \makebox[\linewidth][\@tempA]%
34  {\box\widebox}%
35  \ignorespaces}

```

Come si vede nelle righe 29-34, l'ambiente compone la tabella sporgente dal lato opportuno, ma

quando la tabella è troppo larga si limita a emettere un avviso; in questo caso basta togliere la `W` dal nome dell'ambiente nel file sorgente dove si era introdotto il codice per la tabella larga; si aggiunge anche un asterisco all'ambiente `table` circostante, che diventa l'ambiente `table*`, e la tabella, troppo larga per una colonna e il suo margine, diventa tranquillamente una tabella a piena pagina senza bisogno di apportare correzioni importanti. Si noti per altro che la tabella sporgente, grazie al suo ambiente `Wtabular` può sporgere nel margine esterno anche se la tabella viene composta a piena pagina mediante l'ambiente `table*`.

Si noti ancora che l'etichetta interna generata con la costruzione sintattica delle righe 14-15 è costituita dal prefisso `dzt@` seguito dal numero che l'ambiente ha creato aggiungendo l'unità (`\@ne`) al contatore `TEX \t@bella`; è importante che questo aumento di una unità sia globale, in modo che ogni successiva tabella larga, dovunque si trovi lungo tutta la serie di file che costituiscono il dizionario, abbia il suo numero univoco, affinché l'etichetta sia univoca; per questo è specificata la dichiarazione `\global`. Questo non sarebbe stato necessario se si fosse definito un contatore `LATEX`, diciamo `t@bella`, perché con quello si sarebbe potuto specificare l'aumento di una unità mediante l'espressione:

```
\stepcounter{t@bella}
```

che in `LATEX` produce l'incremento unitario in modo globale. Il contatore `LATEX` avrebbe richiesto una programmazione leggermente più complessa negli altri punti della definizione dell'ambiente, ed è per questo che abbiamo scelto un contatore `TEX`.

4 Altre considerazioni

La classe `dictionarySCR` presenta molte altre caratteristiche; per ragioni di brevità qui ne abbiamo esposte e descritte solo alcune, secondo noi le più importanti.

Ma ci sono anche mezzi per “strutturare” lemmi con descrizioni lunghe creando loro una specie di mini-indice iniziale e dei comandi per i titolini relativi alle sezioni in cui è strutturato il lemma in questione. Ci sono modi di inserire medaglioni con sfondo colorato; per comporre i cataloghi delle stelle in modo che le righe della tabella, piuttosto larghe perché hanno un decina di colonne, siano alternativamente su uno sfondo leggermente colorato oppure bianco, al fine di aiutare l'occhio a restare sulla riga giusta; sullo schermo l'effetto è gradevole; stampato a colori su carta è altrettanto gradevole, ma in tonalità di grigio le ombreggiature di sottofondo forse sono troppo chiare. Le figure in colonna possono avere didascalie sopra o sotto; ogni didascalia presenta una grossa punta di freccia diretta verso la figura a cui la didascalia si riferisce.

Si possono comporre di default testi e citazioni in diverse lingue che includono oltre all'italiano, l'inglese, il francese, il tedesco, il latino e il greco classico; la lingua principale è specificata nelle opzioni di `\documentclass`. A seconda della lingua specificata, il comando `\ofentry` che compare nella definizione di `\eqref` verrà modificato automaticamente in corrispondenza alla lingua scelta: "sub lemma", "sub entry", "sous l'article", "unten das Stichwort", e anche le traduzioni latina "sub lemmate" e greca "ὑπὸ τὸν εἰσὸδον". Queste due ultime traduzioni sono evidentemente delle forzature, visto che non immaginiamo che venga composto un dizionario in latino o in greco classico, ma l'abbiamo fatto per simmetria e per coerenza; e poi non si sa mai...

Se si vogliono inserire citazioni in lingue scritte con ideogrammi bisogna compilare con XeLaTeX usando la classe `xdictionarySCR`, avendo cura di specificare nel preambolo i font giusti e in particolare l'associazione lingua-font; per esempio, una riga di ideogrammi giapponesi può venire semplicemente scritta dentro un ambiente `japanese` (da definire da parte dello scrittore), cosicché il programma faccia tutto da solo. La difficoltà, caso mai, non è tanto nel comporre, quanto nel conoscere la lingua orientale da scrivere, il giapponese nell'esempio specifico; inoltre l'immissione di caratteri katakana, hiragana e kanji con una tastiera occidentale non è la cosa più facile di questo mondo; il sistema operativo Mac OS X delle macchine Macintosh consente di usare una tastiera virtuale e per il giapponese offre un metodo particolare di scrittura; l'editor `emacs` (forse anche l'editor `Aquamacs`) consente una modalità particolare di immissione degli ideogrammi orientali, ma il problema resta e non può essere risolto con la classe. Certo si può provare ad eseguire la tecnica del copia e incolla da altri testi, ma è possibile che il metodo fallisca se non c'è una perfetta corrispondenza fra gli ideogrammi contenuti nel file da cui si copia con gli ideogrammi contenuti nel font usato per comporre il documento dove si incolla. Questi, però, sono problemi che lasciamo volentieri a coloro che conoscono il giapponese e che desiderano scrivere un dizionario che contenga parti in questa lingua.

Naturalmente non è una cosa straordinaria che ci siano stati dei problemi quando si affronta un progetto così complesso. In realtà non è molto complesso costruire e usare una classe per comporre un dizionario; è complesso comporre il *Dizionario di Astronomia* a causa delle lingue, degli alfabeti, delle informazioni strutturate sotto forma di equazioni, tabelle, e simili. Uno dei due autori ha cominciato a scrivere un *Dizionario di Termini Marinari* nel comporre il quale non incontra le difficoltà del primo dizionario, grazie al fatto che in questo secondo non è necessario ricorrere a molta informazione fortemente strutturata.

Ecco perché la classe è ancora al livello "alpha". Anche durante la stesura di questo articolo abbiamo avuto necessità di fare qualche piccolo aggiustamento di "fine tuning" di alcune macro che presentavano ancora alcuni difettucci nati, per esempio, da aggiornamenti della distribuzione TeX Live alla versione del 2012: un caso tipico è stato l'aggiornamento di `hyperref`, che ha richiesto di modificare qualche nostra macro. Anche se ormai la cosa succede molto raramente (e nel secondo dizionario non è successo mai), talvolta i mark delle testatine non corrispondono a quello a cui dovrebbero corrispondere. Nonostante tutte le cure, è proprio il concetto stesso di dizionario a due colonne che talvolta mette in crisi il programma di composizione. Abbiamo messo in opera tutti i *patch* di cui siamo stati capaci, ma in qualche raro caso nel *Dizionario di Astronomia* presenta qualche problema dovuto all'algoritmo asincrono che usano i programmi di composizione per spezzare il testo in colonne e predisporre la pagine per il file di uscita.

Questo processo asincrono produce anche delle nuove rigidzze nel sistemare le numerose figure e tabelle e abbiamo cercato di trovare il modo più delicato possibile per convincere il programma a non lasciare colonne mozze. Tutto sommato siamo abbastanza soddisfatti e abbiamo anche la gratificazione che il nostro lavoro abbia potuto essere usato per comporre altri dizionari di cui uno in una disciplina come lo scintoismo, che non ha nulla a che vedere con il dizionario tematico da cui eravamo partiti.

Elenchiamo molto succintamente alcuni degli altri piccoli problemi che abbiamo incontrato: il problema della impostazione corretta dei comandi per scrivere in greco, che inizialmente, senza voler attivare tutti i marchingegni di `babel`, avevamo ridefinito completamente per poi ricorrere a soluzioni trovate sul Forum GIr (grazie a tutti coloro che hanno usato il Forum sia per domandare sia per rispondere); la colorazione di background delle tabelle; le tabelle mobili poco sporgenti nel margine esterno, di cui abbiamo creato diversi comandi o ambienti, non solo quello descritto qui; le didascalie da inserire sopra o sotto gli oggetti mobili a cui si riferiscono, ma indipendentemente dal tipo di oggetto; la gestione del font di default in modo da poter usare un comando equivalente a `\HUGE`, che non fa parte dei comandi predefiniti né nelle classi standard, né nelle classi `Koma Script`; nonostante un certo numero di lingue fossero precaricate, la loro corretta gestione in modo che fosse il compositore a decidere quale fosse la lingua principale senza dover editare la classe stessa. In un modo o nell'altro li abbiamo risolti tutti.

Attualmente stiamo lavorando all'unione delle due classi, quella per pdfLaTeX e quella per XeLaTeX , in modo che sia quest'unica nuova classe a decidere quali pacchetti caricare e quali macro

usare a seconda del motore di composizione; la lingua greca in questo caso è il problema maggiore; con pdfL^AT_EX si possono usare facilmente molte delle strutture già usate in precedenza, ma bisogna aggiungere la possibilità di usare la codifica utf8 per poter eventualmente introdurre il testo direttamente in greco; con X_YL^AT_EX non ci sono problemi a introdurre direttamente il testo greco, ma bisogna anche adattare al dizionario la stessa traslitterazione in lettere latine corrispondente alla codifica LGR; oggi si dispone di pacchetti adatti a questo scopo, ma bisogna eseguire degli adattamenti per il (nostro) dizionario e la nostra classe.

Riferimenti bibliografici

In questa lista di riferimenti bibliografici \$TEXMF indica la cartella radice sul proprio hard disk dell'albero della distribuzione del sistema T_EX.

ARSENAU, D. (2003). «The wrapfig package». URL \$TEXMF/doc/latex/wrapfig/wrapfig-doc.pdf.

BECCARI, C. e FLECK, H. (2010). «I mark, questi sconosciuti». *ArsT_EXnica*, (9).

BECCARI, C. e GREGORIO, E. (2012). «The

Package imakeidx». URL \$TEXMF/doc/latex/imakeidx/imakeidx.pdf.

KOHM, M. e MORAWSKI, J.-U. (2012). «Koma Script – The Guide». URL \$TEXMF/doc/latex/koma-script/scrguien.pdf.

LEHMAN, P. (2011). «The etoolbox package». URL \$TEXMF/doc/latex/etoolbox/etoolbox.pdf.

VAN OOSTRUM, P. (2004). «Page layout in L^AT_EX». URL \$TEXMF/doc/latex/fancyhdr/fancyhdr.pdf.

WILSON, P. (2011). «The Memoir Class for Configurable Typesetting – User Guide». URL \$TEXMF/doc/latex/memoir/memman.pdf. Documentazione aggiornata da Lars Madsen.

- ▷ Claudio Beccari
Villarbasce (TO)
claudio dot beccari at gmail
dot com
- ▷ Heinrich Fleck
Todi (PG)
heinrich dot fleck at yahoo
dot it

L^AT_EX3: un nuovo gioco per i maghi e per diventarlo

Enrico Gregorio

Sommario

Esaminiamo con esempi alcune delle funzioni messe a disposizione dal nucleo di L^AT_EX3. In particolare ci concentreremo sull'uso delle variabili di tipo *sequence* e faremo un cenno sull'uso del modulo `l3keys`.

Sommario

We shall examine, by means of examples, some of the functions provided by the L^AT_EX3 kernel. In particular, we shall focus on the usage of the *sequence* variable type and show something about the `l3keys` module.

1 Introduzione

Lo sviluppo di L^AT_EX3 sta procedendo piuttosto bene, sebbene sia ancora limitato allo strato di programmazione. Tuttavia la lista dei pacchetti che già lo usano è lunga: `fontspec`, `siunitx` fra i più noti.

Qual è lo scopo del *L^AT_EX3 team*? Lo scopo finale è, naturalmente, sviluppare la versione 3 di L^AT_EX. Ma l'indirizzo attualmente preminente è di fornire un ambiente di programmazione unitario e coerente. La versione sperimentale, su cui sono basati i pacchetti citati, è disponibile anche all'utente comune con

```
\usepackage{expl3}
```

e l'ambiente speciale di programmazione si attiva e disattiva con

```
\ExplSyntaxOn
\ExplSyntaxOff
```

che possiamo immaginare analoghi della coppia formata da `\makeatletter` e `\makeatother`. Si veda GREGORIO (2012) per una rapida introduzione al nuovo linguaggio.

Lo scopo di questo articolo non è di descrivere il linguaggio, ma di farne vedere qualche potenzialità e beneficio.

In tutti gli esempi supporremo di essere nell'ambiente di programmazione dove, è importante saperlo, gli spazi e i fine riga sono ignorati. Questo evita quasi tutti gli inghippi del vecchio tipo di programmazione in cui gli spazi spuri erano e sono un incubo dell'autore di pacchetti, ma anche dell'utente finale.

2 Definire nuovi comandi a livello utente

Il pacchetto che provvisoriamente fornisce l'infrastruttura per definire comandi a livello utente è `xparse`. Facciamo un esempio che ci guiderà per un po' alla scoperta delle nuove funzionalità.

Nello studio dell'algebra dei gruppi le permutazioni sono uno degli strumenti fondamentali, del resto è studiandole che la teoria dei gruppi si è sviluppata. Un metodo per denotare le permutazioni è molto interessante da un punto di vista teorico, perché il *nome* della funzione contiene anche la regola per calcolarne i valori. Questo si ottiene con la cosiddetta *decomposizione in cicli disgiunti*. Per esempio, la permutazione

$$1 \mapsto 5, \quad 2 \mapsto 3, \quad 3 \mapsto 2, \quad 4 \mapsto 1, \quad 5 \mapsto 4$$

si può denotare con

$$(1\,5\,4)(2\,3)$$

e, per sapere chi associare a un elemento basta vedere quale sta alla sua destra (se c'è una parentesi si torna al primo elemento della lista).

Ovviamente, scrivere un ciclo come $(1\,2\,3\,4\,5\,6\,7\,8)$ con il codice

```
(1\,2\,3\,4\,5\,6\,7\,8)
```

è noioso, ma non solo: potremmo voler cambiare i delimitatori o ciò che va fra un elemento e l'altro. Ecco un modo:

```
\NewDocumentCommand{\cycle}
{>\SplitList{,}}m }
{(\ProcessList{#1}{\cycleargument})}
```

```
\newcommand{\maybespace}{%
  \renewcommand{\maybespace}{\,}%
  \newcommand{\cycleargument}[1]{%
    \maybespace#1}
```

Il comando `\cycle` ha un argomento che verrà automaticamente suddiviso nelle parti delimitate da virgole; ciascuna parte verrà passata al secondo argomento di `\ProcessList`. Quindi l'effetto è, alla fine, di eseguire

```
\cycleargument{1}
\cycleargument{2}
\cycleargument{3}
\cycleargument{4}
\cycleargument{5}
```

e il trucco ben noto di ridefinire `\maybespace` inserirà `\`, solo fra gli elementi dopo il primo. Ecco il risultato:

`\cycle{1,2,3,4,5} → (1 2 3 4 5)`

Si può fare meglio? Certo che si può, ma occorrono strumenti un po' più raffinati di `\ProcessList`.

Il nucleo di LATEX3 fornisce decine di funzioni che astraggono compiti ripetitivi, di cui `\SplitList` e `\ProcessList` sono solo la superficie. Al di sotto sta una nuova *struttura di dati*, cioè le variabili di tipo *sequence*.¹

Per poterle usare occorre però entrare nell'ambiente di programmazione di LATEX3 che, al momento, richiede di racchiudere il codice fra

```
\ExplSyntaxOn
...
\ExplSyntaxOff
```

dove un nuovo mondo si apre. Vediamo l'esempio.

```
\NewDocumentCommand{\cycle}{m}
{
  \perms_print_cycle:n { #1 }
}
\seq_new:N \l_perms_cycle_seq
\cs_new_protected:Npn
\perms_print_cycle:n #1
{
  \seq_set_split:Nnn
    \l_perms_cycle_seq { , } { #1 }
  (
    \seq_use:Nnnn \l_perms_cycle_seq
      { \, } { \, } { \, }
  )
}
```

La funzione `\seq_use:Nnnn` dice che fare della *sequence* data come primo argomento: i tre argomenti successivi dicono che cosa mettere fra i vari oggetti che compongono la *sequence*; rispettivamente se ce ne sono solo due, fra i primi $n - 1$ se ce ne sono $n > 2$ e fra gli ultimi due in questo caso. La funzione nasce per risolvere il noto problema “apples and pears” o “apples, pears, and oranges” come vuole un certo stile di scrittura diffuso principalmente negli USA.

Vediamo passo passo. La prima dichiarazione traduce `\cycle` in un comando interno, come raccomandano le linee guida se l'esecuzione non può essere ridotta a comandi “esterni” come nel caso precedente. Meglio parlare di *funzioni* che di comandi interni: sono *funzioni* tutti quei comandi il cui nome contiene i due punti, dopo i quali vanno indicati gli argomenti della funzione, in questo caso uno. Prima di definire la nostra funzione dichiariamo la variabile *sequence* che useremo: le linee guida specificano che ogni variabile deve essere

1. Preferisco non tradurre i nomi dei tipi di variabili.

dichiarata. Le funzioni di tipo `set` ne stabiliscono un valore.

La funzione `\perms_print_cycle:n` è definita come `protected` perché a sua volta esegue funzioni dello stesso tipo (lo sono tutte quelle di tipo `set`). Ha un argomento di tipo normale, cioè espresso tra graffe. Il suo primo compito è di assegnare il valore alla variabile spezzando l'argomento alle virgole; il secondo è di stampare via via gli elementi della *sequence* separati da `\,`, mettendo in testa e in coda le parentesi.

Vediamo il risultato con questo esempio

`\cycle{1,2,3,4,5} → (1 2 3 4 5)`

che funziona, ma ci lascia ancora insoddisfatti. Vorremmo poter cambiare il delimitatore fra gli elementi del ciclo: alcuni preferiscono la virgola, per esempio. Così sperimentiamo il codice seguente

```
\NewDocumentCommand{\cycle}{O{\,} m}
{
  \perms_print_cycle:nn { #1 } { #2 }
}
\seq_new:N \l_perms_cycle_seq
\cs_new_protected:Npn
\perms_print_cycle:nn #1 #2
{
  \seq_set_split:Nnn \l_perms_cycle_seq
    { , } { #2 }
  (
    \seq_use:Nnnn \l_perms_cycle_seq
      { #1 } { #1 } { #1 }
  )
}
```

dove specifichiamo che `\cycle` ha un primo argomento facoltativo, il cui valore normale è `\,`; così avremo

`\cycle{1,2,3,4,5} → (1 2 3 4 5)`
`\cycle[,]{1,2,3,4,5} → (1,2,3,4,5)`

In realtà la definizione ufficiale che uso è nel codice 1. La funzione `\egreg_seq_use:Nn` evita di dover scrivere tre volte lo stesso argomento; ha un prefisso diverso perché la uso in varie situazioni (in un'applicazione reale farebbe parte di un pacchetto). Ma perché definire la funzione interna con quattro argomenti invece che con due? Perché possiamo fare molto di più:

```
\NewDocumentCommand{\pcycle}{O{\,} m}
{
  \perms_print_cycle:nnnn
    { ( } { ) } { #1 } { #2 }
}
\NewDocumentCommand{\bcycle}{O{\,} m}
{
  \perms_print_cycle:nnnn
    { [ ] { ] } { #1 } { #2 }
}
```

```
\NewDocumentCommand{\cycle}{0{\,} m}
{
  \perms_print_cycle:nnnn { ( } { ) } { #1 } { #2 }
}

\cs_new:Npn \perms_seq_use:Nn #1 #2
{
  \seq_use:Nnnn #1 { #2 } { #2 } { #2 }
}
\seq_new:N \l_perms_cycle_seq
\cs_new_protected:Npn \perms_print_cycle:nnnn #1 #2 #3 #4
{
  \seq_set_split:Nnn \l_perms_cycle_seq { , } { #4 }
  #1
  \egreg_seq_use:Nn \l_perms_cycle_seq { #3 }
  #2
}
```

CODICE 1: Versione ufficiale per `\cycle`

```
\NewDocumentCommand{\vcycle}{0{\,} m}
{
  \perms_print_cycle:nnnn
  { \lvert } { \rvert } { #1 } { #2 }
}
```

Il tutto senza dover definire nuove funzioni interne. Vedremo più avanti che si può fare ancora meglio.

3 Cicli dentro cicli

Una notazione comune per le matrici in certi linguaggi come MATLAB è

`{{1,2,3},{4,5,6}}`

per denotare la matrice

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

senza dover introdurre le due dimensioni. Ci sembra un ottimo candidato per sfruttare le variabili *sequence*.

Qui c'è una piccola complicazione: siccome dobbiamo costruire una tabella con `&` e `\\`, dovremo operare in due passi. Il primo è di memorizzare tutto il materiale in una variabile di tipo *token list* e poi liberarne il contenuto nell'ambiente *array*.

Una delle funzioni importanti al proposito è `\seq_map_inline:Nn` che agisce sulla *sequence* data come primo argomento, ripetendo il codice dato come secondo argomento per ogni elemento della *sequence*. Tanto per fare un esempio

```
\seq_clear:N \l_tmpa_seq
\seq_put_right:Nn \l_tmpa_seq {a}
\seq_put_right:Nn \l_tmpa_seq {b}
\seq_put_right:Nn \l_tmpa_seq {x}
```

prima azzerata la *sequence* data e poi aggiunge i tre elementi. Si noti che dobbiamo prima 'svuotare' la

variabile `\l_tmpa_seq` perché le operazioni di tipo `put` aggiungono a ciò che c'è già. Ora, con

```
\fbox{
  \seq_map_inline:Nn \l_tmpa_seq { (#1) }
}
```

otterremmo $\boxed{(a)(b)(x)}$.

Il nostro scopo è di definire un comando `\matlabmatrix` che prenda la notazione MATLAB e la trasformi in una matrice vera. Vediamo che cosa ci potrebbe servire: un argomento facoltativo che ci indichi quali delimitatori vogliamo e la matrice stessa.

```
\NewDocumentCommand{\matlabmatrix}
{ 0{b} m }
{
  \matlabmatr_print:nn { #1 } { #2 }
}
```

Finora è facile, no? Ogni comando a livello utente dovrebbe avere una sua versione interna senza argomenti facoltativi, ma solo normali. La `b` sarà usata per comporre l'ambiente `bmatrix`, potremo dare anche `p` o `v`.

Il primo passo è di memorizzare la lista esterna in una *sequence*:

```
\seq_set_split:Nnn
\l_matlabmatr_rows_seq
{ , } { #2 }
```

e poi applicare un *mapping*

```
\seq_map_inline:Nn
\l_matlabmatr_rows_seq
{
  \matlabmatr_makerow:n { ##1 }
}
```

e, per finire, comporre la matrice:

```
\begin{#1matrix}
\l_matlabmatr_matrix_tl
\end{#1matrix}
```

dove useremo la *token list* che conterrà la matrice nella forma con `&` e `\\`; dobbiamo ricordarci di svuotarla del suo valore prima di operare. Ci occuperemo più avanti della funzione che costruisce le righe; intanto ecco il codice per `\matlabmatr_print:nn` tutto insieme:

```
\seq_new:N \l_matlabmatr_rows_seq
\tl_new:N \l_matlabmatr_matrix_tl
\cs_new_protected:Npn
\matlabmatr_print:nn #1 #2
{
\tl_clear:N \l_matlabmatr_matrix_tl
\seq_set_split:Nnn
\l_matlabmatr_rows_seq
{ , } { #2 }
\seq_map_inline:Nn
\l_matlabmatr_rows_seq
{
\matlabmatr_makerow:n { ##1 }
}
\begin{#1matrix}
\l_matlabmatr_matrix_tl
\end{#1matrix}
}
```

Perché `##1`? Perché con `#1` nel codice di prima si denota l'elemento della *sequence* che viene passato alla funzione di *mapping* e qui siamo all'interno di una definizione. Abbiamo anche allocato una nuova variabile *token list* e la azzeriamo prima di cominciare il lavoro.

Ora la nostra funzione che costruisce le righe è già nota: abbiamo risolto lo stesso problema con i cicli. Qui c'è una differenza: non vogliamo stampare subito gli elementi, ma memorizzare la matrice nella *token list*.

```
\seq_new:N \l_matlabmatr_arow_seq
\cs_new_protected:Npn
\matlabmatr_makerow:n #1
{
\seq_set_split:Nnn
\l_matlabmatr_arow_seq
{,} { #1 }
\tl_put_right:Nx
\l_matlabmatr_matrix_tl
{
\egreg_seq_use:Nn
\l_matlabmatr_arow_seq
{ & }
}
\tl_put_right:Nn
\l_matlabmatr_matrix_tl
{ \\ }
}
```

La funzione `\tl_put_right:Nn` accoda qualcosa a quanto già memorizzato nella *token list*: inseriamo

`&` fra i vari elementi e un `\\` alla fine. La usiamo anche nella forma `\tl_put_right:Nx` in modo che non venga accodato

```
\egreg_seq_use:Nn
\l_matlabmatr_arow_seq { & }
```

ma il suo sviluppo usando il valore attuale della *sequence*. Se consultiamo il manuale di LATEX3 (THE LATEX3 TEAM, 2012) scopriamo che accanto a `\seq_use:Nnnn` c'è una stellina: dice della funzione che è *espandibile*, cioè può essere usata dove l'argomento è denotato con `x` dando il risultato sperato. Le funzioni dichiarate come *protected* non sono *mai* espandibili e, infatti, `\egreg_seq_use:Nn` è stata definita con il semplice `\cs_new:Npn` per non perdere l'espandibilità nel modo abbastanza ovvio

```
\cs_new:Npn \egreg_seq_use:Nn #1 #2
{
\seq_use:Nnnn #1 { #2 } { #2 } { #2 }
}
```

Il discorso sull'espandibilità è un po' troppo tecnico e ci porterebbe troppo lontano. Ciò che conta è che il comando funziona!

```
\matlabmatrix{{1,2,3},{4,5,6}}
```

$$\downarrow$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Proviamo con l'argomento facoltativo:

```
\matlabmatrix[v]{{1,2,3},{4,5,6}}
```

$$\downarrow$$

$$\left| \begin{array}{ccc} 1 & 2 & 3 \\ 4 & 5 & 6 \end{array} \right|$$

4 Un'applicazione frivola

Tanto per scoprire nuove funzioni, vediamo come trasformare una lista di numeri interi separati da virgole nella loro somma. Per rendere l'argomento un po' più interessante, scriveremo anche la somma in colonna: si veda il codice 2. Vediamo, come al solito, che funziona:

$$\begin{array}{r} 11 + \\ 2 + \\ 43 + \\ 84 = \\ \hline \end{array}$$

$$\backslash colsum\{11,2,43,84\} \rightarrow 140$$

Qui non c'è molto di nuovo, se non il fatto che possiamo usare una *sequence* quante volte vogliamo e in modi diversi; nel secondo impiego la diamo in pasto a `\int_eval:n` che si occupa di eseguire la somma.

E se volessimo scrivere una moltiplicazione in colonna? Si guardi il codice 3.

Qui dobbiamo solo ricordarci di rovesciare la *sequence*, perché l'operazione si esegue da destra

```
\NewDocumentCommand{\colsum}{m}
{
  \colsum_do:n { #1 }
}
\tl_new:N \l_colsum_list_tl
\seq_new:N \l_colsum_list_seq
\cs_new_protected:Npn \colsum_do:n #1
{
  \tl_clear:N \l_colsum_list_tl
  \seq_set_split:Nnn \l_colsum_list_seq { , } { #1 }
  \tl_put_right:Nx \l_colsum_list_tl
    { \egreg_seq_use:Nn \l_colsum_list_seq { & + \\ } }
  \tl_put_right:Nn \l_colsum_list_tl { & = \\ \midrule}
  \tl_put_right:Nx \l_colsum_list_tl
    {
      \int_eval:n { \egreg_seq_use:Nn \l_colsum_list_seq { + } }
    }
  \begin{array}{r@{\;}l}
    \l_colsum_list_tl
  \end{array}
}
```

CODICE 2: Somma in colonna

```
\NewDocumentCommand{\colmult}{mm}
{
  \colmult_do:nn { #1 } { #2 }
}
\tl_new:N \l_colmult_spaces_tl
\tl_new:N \l_colmult_partial_summands_tl
\seq_new:N \l_colmult_multiplier_seq
\cs_new_protected:Npn \colmult_do:nn #1 #2
{
  \tl_clear:N \l_colmult_partial_summands_tl
  \tl_clear:N \l_colmult_spaces_tl
  \seq_set_split:Nnn \l_colmult_multiplier_seq { } { #2 }
  \seq_reverse:N \l_colmult_multiplier_seq
  \seq_map_inline:Nn \l_colmult_multiplier_seq
    {
      \tl_put_right:Nx \l_colmult_partial_summands_tl
        {
          \int_eval:n { #1 * ##1 }
        }
      \tl_put_right:NV \l_colmult_partial_summands_tl \l_colmult_spaces_tl
      \tl_put_right:Nn \l_colmult_partial_summands_tl { \\ }
      \tl_put_right:Nn \l_colmult_spaces_tl { \hphantom{0} }
    }
  \begin{array}{r@{\;}l}
    #1 & \times \\
    #2 & = \\
    \midrule
    \l_colmult_partial_summands_tl
    \midrule
    \int_eval:n { #1 * #2 }
  \end{array}
}
```

CODICE 3: Moltiplicazione in colonna

a sinistra; la funzione si chiama, in modo piuttosto intuitivo, `\seq_reverse:N`. Vediamo anche che un argomento può essere spezzato “a niente”: così il numero 294 diventa la *sequence* `{2}{9}{4}`. Vediamo l'esempio:

$$\begin{array}{r} 412 \times \\ 294 = \\ \hline 1648 \\ 3708 \\ 824 \\ \hline \end{array}$$

`\colmult{412}{294} → 121128`

A ogni passo del *mapping* aggiungiamo alla *token list* ausiliaria un `\hphantom{0}` per mantenere il corretto allineamento.

Meglio nemmeno provare a scrivere una cosa del genere per $\text{\LaTeX} 2_{\epsilon}$: ci si perderebbe la testa in poco tempo: dovremmo sistemare la ricorsività, invertire il moltiplicatore, scrivere le macro per i calcoli. Invece la macro `\colmult` è stata scritta in circa cinque minuti, compresa la correzione degli errori delle versioni iniziali.

5 Altre interfacce

Il modo con cui abbiamo scritto `\perms_print_cycle:nnnn` ci permette di definire nuove interfacce: per esempio

```
\xcycle[delims=(),
separator=\,,]
{1,2,3,4,5}
\xcycle[delims={[]},
separator={;},]
{1,2,3,4,5}
```

ma senza dover ridefinire il comando interno. Vediamo come:

```
\NewDocumentCommand{\xcycle}{0{} m}
{
  \keys_set:nn {perms}
  {delims=(),separator=\,,}
  \keys_set:nn {perms} { #1 }
  \perms_print_cycle:VVVn
  \l_perms_left
  \l_perms_right
  \l_perms_separator
  { #2 }
}
\cs_generate_variant:Nn
\perms_print_cycle:nnnn
{VVV}
\l_new:N \l_perms_left
\l_new:N \l_perms_right
\l_new:N \l_perms_separator
```

La funzione `\perms_print_cycle:VVVn` è definita in termini di `\perms_print_cycle:nnnn` dicendo che i primi tre argomenti devono essere presi come il valore delle variabili indicate.

Si tratta ora di associare i valori. Ci occupiamo prima di assegnare i valori normali e poi, eventualmente, di cambiarli se viene espresso l'argomento facoltativo.

```
\keys_define:nn { perms }
{
  delims .code:n = \perms_set_delims:NN #1,
  separator .tl_set:N = \l_perms_separator,
}
\cs_new_protected:Npn
\perms_set_delims:NN #1 #2
{
  \tl_set:N \l_perms_left #1
  \tl_set:N \l_perms_right #2
}
```

Una dichiarazione di chiave, cioè data in `\keys_define:nn`, come

```
delims .code:n = \perms_set_delims:NN #1,
```

dice che il valore a destra dell'uguale dopo la chiave `delims`, rappresentato da `#1`, deve essere passato alla funzione `\perms_set_delims:NN`. Invece

```
separator .tl_set:N = \l_perms_separator,
```

è una forma abbreviata di

```
separator .code:n =
\l_set:N \l_perms_separator { #1 },
```

Vediamo le possibili combinazioni:

$$\begin{array}{r} \xcycle\{1,2,3\} \\ (1\,2\,3) \\ \hline \xcycle[delims={[]}]{1,2,3} \\ [1\,2\,3] \\ \hline \xcycle[separator=;]{1,2,3} \\ (1;2;3) \\ \hline \xcycle[delims={},separator=;]{1,2,3} \\ [1;2;3] \end{array}$$

Per fare un esempio un po' più reale, si pensi al comando `\parbox` che accetta ben tre argomenti facoltativi:

```
\parbox[⟨pos⟩][⟨height⟩][⟨innerpos⟩]{⟨text⟩}
```

e ci si dimentica sempre quali siano. Una sintassi del tipo

```
\xparbox[pos=⟨pos⟩,
height=⟨height⟩,
innerpos=⟨innerpos⟩]{⟨text⟩}
```

oppure una sintassi per `\chapter` del tipo

```
\xchapter[numbered=⟨boolean⟩,
toc=⟨text⟩,
header=⟨text⟩]{⟨text⟩}
```

sarebbero certamente le benvenute. Non è in realtà troppo difficile scrivere `\xchapter` in termini del comando attuale, si veda il codice 4. Ovviamente definire `\xchapter` in termini di `\chapter` esegue parecchi compiti più volte; ma la potenza del nuovo nucleo è evidente.

```

\NewDocumentCommand{\xchapter}{ 0{} m }
{
  \keys_set:nn {chapter}
  {
    numbered=true,
    toc=#2,
    head=#2,
    #1
  }
  \class_xchapter:Vn \l_class_toctitle_tl \l_class_headtitle_tl { #2 }
}
\bool_new:N \l_chapter_numbered_bool
\tl_new:N \l_chapter_toctitle_tl
\tl_new:N \l_chapter_headtitle_tl
\keys_define:nn {chapter}
{
  numbered .bool_set:N = \l_chapter_numbered_bool,
  toc .tl_set:N = \l_chapter_toctitle_tl,
  head .tl_set:N = \l_chapter_headtitle_tl,
}
\cs_new_protected:Npn \class_xchapter:nnn #1 #2 #3
{
  \bool_if:NTF \l_chapter_numbered_bool
  {
    \chapter[#1]{#3}\markboth{\MakeUppercase{#2}}{#3}
  }
  {
    \chapter*{#3}
  }
}
\cs_generate_variant:Nn \class_xchapter:nnn {VV}

```

CODICE 4: Il comando `\xchapter`

Riferimenti bibliografici

GREGORIO, E. (2012). «Scrivere un pacchetto per LATEX3 – il pacchetto kantlipsum». *ArsTEXnica*, **13**.

THE LATEX3 TEAM (2012). «The LATEX3 interfaces». Documentation manual, The LATEX3 Project. URL <http://texdoc.net/texmf-dist/doc/latex/l3kernel/interface3.pdf>. Accessibile con texdoc interface3.

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

Un pacchetto per il controllo del codice fiscale italiano

Claudio Beccari

Sommario

Il codice fiscale, che codifica le generalità della persona a cui si riferisce, deve essere generato dall'amministrazione dello Stato, ma tutti necessitano di un software per controllare di averlo scritto in modo corretto; il software deve quindi essere in grado di verificare che i dati siano coerenti, in particolare deve essere in grado di verificare la correttezza della lettera di controllo finale.

Abstract

The Italian Fiscal Code is a special string that encodes the personal identity data; it may be issued only by the State Administrative Offices, but when a document is being written a piece of software should be available so as to check the correctness of the encoded string, in particular it must control the final check letter for coherence with the remaining data.

1 Introduzione

Il *codice fiscale* è stato introdotto in Italia con il Decreto Ministeriale del 23 dicembre 1976 dal titolo "Sistemi di codificazione dei soggetti da iscrivere all'anagrafe tributaria", (MINISTRO DELLE FINANZE, 1976); esso riguarda non solo le persone fisiche ma anche le persone giuridiche e le altre entità da iscrivere all'anagrafe tributaria anche se prive di personalità giuridica.

Qui mi occuperò del codice usato per le persone fisiche, normalmente conosciuto come *codice fiscale* per antonomasia. Tra l'altro il codice per le persone fisiche viene usato anche come identificativo per i servizi sanitari e svolge una funzione simile ai *social security code* e ai *person number* in vigore in molti paesi stranieri.

Tutti abbiamo il nostro codice personale e sappiamo che esso è una stringa alfanumerica di 16 lettere e cifre ed ha la forma:

CCCNNAAMGLLLLK

Non tutti abbiamo chiaro come siano legate quelle lettere o cifre ai nostri dati personali; sostanzialmente:

CCC sono tre lettere estratte dal cognome;

NNN sono tre lettere estratte dal nome;

AA sono le ultime due cifre dell'anno di nascita;

M è una lettera che identifica il mese di nascita;

GG sono due cifre che rappresentano il giorno di nascita, aumentato del valore 40 per le donne, cosicché anche il sesso della persona viene codificato mediante questa aggiunta;

LLLL sono una lettera seguita da tre cifre che codificano il luogo di nascita: il codice del comune per i nati in Italia o il codice della nazione per i nati all'estero;

K è una lettera di controllo che viene determinata in base ai 15 caratteri precedenti; è una specie di "prova del nove" per controllare la correttezza della stringa che forma l'intero codice fiscale.

Per evitare *omocodie*, cioè i rari casi in cui due o più persone hanno i primi quindici caratteri uguali (e quindi anche il sedicesimo carattere di controllo), le sette cifre che compaiono nel codice possono venire sostituite con lettere, secondo regole specificate nel Decreto citato ma, evidentemente, solo l'amministrazione centrale dello Stato può sapere se un dato codice sia già stato assegnato a chi e quando, in modo da poter eseguire le sostituzioni senza generare duplicati.

Spesso capita di scrivere documenti nei quali è necessario riportare il codice fiscale; non sempre si ha a disposizione il tesserino dell'Agenzia delle Entrate o la Tessera Sanitaria; ma anche disponendone capita di commettere degli errori di digitazione.

Il pacchettino che presento qui, da usare con il markup di $\text{\LaTeX}$, serve appunto per verificare che la lettera di controllo K sia conforme con gli altri dati secondo l'algoritmo specificato nel Decreto citato. Questo non assicura che il codice sia corretto in ogni suo dettaglio, perché sono sempre possibili errori che si compensano, ma l'eventualità che questa verifica dia un falso positivo è molto remota.

2 L'algoritmo di verifica

Ad ogni carattere, lettera o cifra, che compare nelle prime 15 posizioni della stringa viene assegnato un valore numerico a seconda della sua posizione nella stringa; si esegue poi la somma di questi valori e se ne calcola il valore modulo 26; cioè si calcola il resto della divisione intera fra questa somma e il numero 26; il valore modulare è un numero compreso fra 0

TABELLA 1: Valori da assegnare ai primi 15 caratteri della stringa del codice fiscale

Carattere	Posizione pari	Posizione dispari
0	0	1
1	1	0
2	2	5
3	3	7
4	4	9
5	5	13
6	6	15
7	7	17
8	8	19
9	9	21
A	0	1
B	1	0
C	2	5
D	3	7
E	4	9
F	5	13
G	6	15
H	7	17
I	8	19
J	9	21
K	10	2
L	11	4
M	12	18
N	13	20
O	14	11
P	15	3
Q	16	6
R	17	8
S	18	12
T	19	14
U	20	16
V	21	10
W	22	22
X	23	25
Y	24	24
Z	25	23

e 25, e serve per identificare la lettera dell’alfabeto “latino” di 26 lettere che occupa quella posizione (A corrisponde a 0, B a 1, ..., Z a 25); questa è la lettera di controllo.

La parte complicata consiste proprio nell’assegnazione del valore ad ogni carattere a seconda della sua posizione: bisogna rifarsi alla tabella 1.

Si noti che la lettera di controllo, occupando la sedicesima posizione, è in posizione pari, quindi anch’essa corrisponde alle indicazioni della tabella 1, solo che nel suo caso se ne conosce il valore e bisogna determinare la lettera.

L’algoritmo non è complicato, ma fare i conti a mano è sempre fonte di possibili errori anche se si eseguono le somme con l’aiuto di una calcolatrice; un programmino *ad hoc* risulta quindi sempre utile.

3 Il pacchetto `codicefiscaleitaliano`

Il seguente pacchetto realizza l’algoritmo di controllo del codice fiscale. Si noti che il motore di composizione, sia esso `pdflatex` o `xelatex`, non è particolarmente adatto per eseguire calcoli; probabilmente `context mkiv` e `lualatex` sono attrezzati meglio, ma la stragrande maggioranza degli utenti usa quasi esclusivamente il primo programma `pdflatex` e alcuni usano `xelatex`; gli altri due programmi citati sono usati prevalentemente da professionisti e da persone in grado di sfruttare appieno il linguaggio di scripting Lua, ma non rappresentano ancora la maggioranza degli utenti del sistema TEX.

I problemi che l’algoritmo presenta sono quelli di determinare il valore da associare ad una lettera a seconda della sua posizione e il passaggio inverso dal risultato dei calcoli alla lettera di controllo; le somme e le altre operazioni fra numeri interi non sono un problema per il motore di composizione che, appunto, è costruito per lavorare *solo* con l’aritmetica intera, tanto che, in altre applicazioni, per eseguire calcoli con i numeri fratti bisogna ricorrere a “sporchi trucchi”, che D.E. Knuth stesso chiama *dirty tricks*. Qui almeno non abbiamo bisogno di sporchi trucchi per eseguire i calcoli, ma ne necessitiamo per eseguire le conversioni dai caratteri ai numeri e viceversa.

Per fare questo sono dovuto scendere a livello di comandi primitivi del sistema TEX; poi ho usato dei costrutti estratti dal nucleo di LATEX; mi sono servito sia delle espressioni numeriche primitive sia di quelle introdotte con l’estensione ϵ -TEX che, ormai da diversi anni, è integrata con tutti i motori di composizione; userò anche i registri numerici, i contatori, identificati con numeri superiori a 255, e mi servirò dei comandi primitivi per assegnare dei nomi a questi contatori, ma non ne allocherò nessuno né con `\newcount` né con `\newcounter`, mantenendo tutto l’algoritmo dentro ad un gruppo, cosicché alla fine ogni contatore e ogni variabile usata riprende il suo valore precedente.

Il codice è il seguente, ma lo commenterò facendo riferimento ai numeri di riga, non necessariamente nell’ordine in cui il codice è scritto.

```

1 \NeedsTeXFormat{LaTeX2e}[2011/06/20]
2 \ProvidesPackage{codicefiscaleitaliano}%
3 [2012/05/06 v.1.2 Controlla il codice
4   fiscale italiano]
5 \newif\ifcontrollo \controllotrue
6 \newif\ifstampacodice \stampacodicefalse
7 \def\getCFletter#1#2!{%
8   \ifx#1\space\getCFletter#2!\else
9   \Letter='#1\def\CFisc{#2}\fi}
10 \def\getOddValore{%
11   \ifnum\Letter<A
12     \valore=\expandafter
13     \ifcase\numexpr\Letter-\zero\relax

```

```

14 1\or0\or5\or7\or9\or13\or15\or17\or19
15 \or21\fi
16 \else
17 \valore=\expandafter
18 \ifcase\numexpr\Letter-\A\relax
19 1\or0\or5\or7\or9\or13\or15\or17\or19%
20 \or21\or2\or4\or18\or20\or11\or3\or6\or8%
21 \or12\or14\or16\or10\or22\or25\or24\or23
22 \fi\fi}
23 \def\getEvenValore{%
24 \ifnum\Letter<\A
25 \valore=\numexpr\Letter-\zero\relax
26 \else
27 \valore=\numexpr\Letter-\A\relax
28 \fi}
29 \newcommand*\codicefiscaleitaliano{%
30 \@ifstar{\c@dfiscit[\stampacodicetrue]}
31 {\c@dfiscit}}
32 \newcommand*\c@dfiscit[2]
33 [\stampacodicefalse]{#1\edef\CFisc{#2}%
34 \let\codfisc\CFisc
35 \begingroup
36 \countdef\cifra=256 \cifra=\z@
37 \countdef\A=258\A='\A
38 \countdef\zero=260 \zero='\0
39 \countdef\Letter=262
40 \countdef\valore=264
41 \countdef\somma=266 \somma=\z@
42 \@whilenum\cifra<16\do{\advance\cifra\@ne
43 \ifx\CFisc\@empty
44 \cifra=16\controllofalse
45 \PackageError{codicefiscaleitaliano}{%
46 \MessageBreak
47 *****
48 \MessageBreak
49 Il codice fiscale #2\MessageBreak
50 non ha 16 caratteri.
51 \MessageBreak
52 L'esecuzione della verifica viene
53 \MessageBreak interrotta.\MessageBreak
54 *****
55 }{Premere <invio> per continuare}%
56 \else
57 \expandafter\getCFletter\CFisc!\relax
58 \ifodd\cifra
59 \getOddValore%
60 \else
61 \getEvenValore%
62 \fi
63 \advance\somma\valore\fi}%
64 \unless\ifx\CFisc\@empty\controllofalse
65 \PackageError{codicefiscaleitaliano}{%
66 \MessageBreak
67 *****
68 \MessageBreak
69 Il codice immesso #2\MessageBreak
70 contiene piu' di 16 caratteri.
71 \MessageBreak
72 L'esecuzione della verifica viene
73 \MessageBreak interrotta.\MessageBreak
74 *****
75 }{Premere <invio> per continuare.}%
76 \else
77 \ifcontrollo
78 \advance\somma-\valore
79 \Letter\somma
80 \divide\Letter by 26\relax
81 \somma=\numexpr\somma - 26*\Letter\relax
82 \ifnum\valore=\somma
83 \PackageInfo{codicefiscaleitaliano}{%
84 \MessageBreak Codice fiscale OK}
85 \else
86 \controllofalse
87 \PackageError{codicefiscaleitaliano}{%
88 \MessageBreak
89 *****
90 \MessageBreak
91 Codice fiscale #2 errato\MessageBreak
92 *****
93 }{Premi S oppure Q oppure <invio>;
94 \MessageBreak
95 il file verra' elaborato lo stesso ma
96 \MessageBreak
97 il codice fiscale deve venire
98 ricontrollato!\MessageBreak
99 *****
100 \fi\fi\fi
101 \ifcontrollo\ifstampacodice\codfisc\fi\fi
102 \endgroup}
103 \endinput

```

Il cuore dell'algoritmo si trova nelle righe 57–63: con l'aiuto del contatore `\cifra`, che punta alla posizione specifica del carattere nella stringa del codice fiscale, si esegue un ciclo ripetitivo mediante `\@whilenum` per esaminare uno alla volta i caratteri del codice; mediante `\getCFletter` si estrae il valore numerico ASCII del carattere estraendolo dalla stringa del codice; questa stringa del codice si accorcia via via e si riduce a una stringa vuota al sedicesimo carattere estratto; questo, però, non influenza il valore del codice che l'utente ha dato in pasto all'algoritmo, se non altro perché il tutto viene eseguito dentro il gruppo aperto con `\begingroup` nella riga 35 e che viene chiuso con `\endgroup` nella riga 102.

In base alla parità del numero contenuto nel contatore `\cifra` vengono eseguite le macro `\getOddValore` oppure `\getEvenValore` che determinano il valore da assegnare al carattere in base alla tabella 1; questo valore viene caricato nel contatore `\valore` e subito dopo si esegue la somma nel contatore `\somma`; si noti che la somma contiene anche l'addendo corrispondente alla lettera di controllo, la sedicesima esaminata. Il valore di questa sedicesima lettera è contenuto nel contatore `\valore` all'ultimo giro, quindi lo si può sottrarre (e lo si farà dopo alcuni controlli nella riga 78) e lo si può usare per controllare se esso ha il valore che

dovrebbe avere se il codice fiscale è giusto, cioè se il suo valore corrisponde alla somma modulo 26 dei primi quindici caratteri.

In effetti nella riga 79 si usa il contatore `\Letter`, non più necessario, ma utile per conservare il quoziente della divisione per 26 troncato per difetto all'intero più vicino; nelle righe 80–81 questo quoziente viene moltiplicato per 26 e il suo risultato viene sottratto dal contatore `\somma` per determinare il resto, cioè il valore modulare, raccolto ancora nel contatore `\somma` il cui precedente valore complessivo non interessa più.

Ecco: a questo punto in `\valore` abbiamo il valore della lettera di controllo associata al codice da controllare e in `\somma` abbiamo il valore modulare che ci consentirebbe di determinare la lettera di controllo giusta; se i valori numerici contenuti in `\valore` e `\somma` sono uguali, i dati immessi per il controllo danno esito positivo, altrimenti viene emesso un messaggio di errore; il codice introdotto non viene modificato, ma l'utente viene avvisato che il codice che ha introdotto è errato e sta a lui scoprire dove ha sbagliato a scrivere il codice; sottolineo che non sarebbe corretto cambiare la lettera di controllo del codice introdotto solo per renderlo coerente con l'algoritmo di verifica; l'errore potrebbe essere altrove e quindi fare una cosa del genere vorrebbe dire introdurre un secondo errore per compensare gli effetti del primo.

Si noti che il comando riservato all'utente può essere asteriscato, come si vede nelle righe 29–31. Il comando senza asterisco semplicemente esegue il controllo; se non ci sono errori, nessuno comando permette di scrivere il codice giusto nel documento che si sta allestendo; se invece si usa il comando asteriscato e le verifiche sono positive, il codice dato in pasto al comando viene trascritto nel documento.

Un altro dettaglio: invece di dare la stringa di sedici lettere tutte di seguito è consentito dare la stringa intercalata da spazi, per renderne più esplicite le sue varie parti ma anche per scriverne il codice completo nel documento che si sta allestendo consentendo, grazie agli spazi, di evitare righe mal composte con una parte del codice fiscale che sporge nel margine (Overfull hbox); questi spazi sono irrilevanti per l'algoritmo di verifica, che semplicemente li salta; si veda più avanti l'esempio d'uso.

Le altre macro definite nelle righe che non ho ancora commentato sono ancillari rispetto all'algoritmo principale. In particolare nelle righe da 36 a 41 si definiscono dei nomi per i contatori numerati con i numeri pari da 256 a 266, superiori a 255, (cosa che sarebbe impossibile fare senza le estensioni di ϵ -T_EX); in realtà non è così importante usare questi contatori aggiuntivi, perché comunque tutto rimane locale dentro il gruppo, ma è una abitudine che io ho e che mi dà la sicurezza di non inter-

ferire nemmeno per sbaglio con i contatori usati normalmente dalle macro del nucleo di L^AT_EX.

Nelle righe 7–8 viene definita la macro ad argomenti delimitati `\getCFletter`: essa riceve la stringa del codice fiscale eventualmente accorciata da prelievi precedenti; la stringa viene terminata con un punto esclamativo nella chiamata; il primo token della stringa, cioè il primo carattere, viene prelevato come argomento #1, mentre tutti gli altri vengono assegnati all'argomento #2 che poi la macro usa per ridefinire la nuova stringa accorciata dal prelievo del primo carattere; si noti che se questo primo carattere è uno spazio la macro richiama se stessa, in modo da non tenere conto dello spazio; questo consente di intercalare degli spazi nella stringa alfanumerica di 16 caratteri del codice in modo da separarne visivamente le sue componenti.

Ma l'assegnazione `\Letter='#1` costituisce lo “sporco trucco” con il quale il codice ASCII del carattere che costituisce il primo argomento viene assegnato al contatore `\Letter`. Normalmente, e lo si è fatto nelle assegnazioni delle righe 37 e 38, non si usa il backtick da solo ma si usa la sequenza `'\` per estrarre il codice ASCII di un carattere. Se si fosse scritto `\Letter='\#1`, il contatore `\Letter` sarebbe stato caricato con il codice ASCII del segno #, non dell'argomento #1.

Nelle righe da 23 a 28 viene definita la macro per assegnare il valore ai caratteri di posto pari: per le cifre basta prendere il codice della cifra e sottrargli il codice dello zero; per le lettere basta prendere il codice della lettera e sottrargli il codice della lettera 'A'; cosicché le assegnazioni del valore ai caratteri di posto pari è decisamente semplice, come per altro è evidente anche dalla seconda colonna della tabella 1.

Invece dalla riga 10 alla riga 22 si trova la definizione della macro `\getOddValore`, molto più complessa, perché in base al codice ASCII della lettera da esaminare a cui bisogna sottrarre il codice di zero o di 'A' a seconda che sia una cifra o una lettera, si può generare un puntatore per far lavorare il costrutto `\ifcase...\or...\or...\fi` al fine di estrarre il valore giusto dalla tabella corrispondente alla terza colonna della tabella 1.

Diversi test vengono eseguiti e oltre al messaggio relativo alla inesattezza del codice, altri due testi informano se il codice immesso è più corto o più lungo di 16 caratteri.

4 Commenti

Il pacchettino è stato controllato nel senso che ho controllato diversi codici fiscali; per esempio, dopo aver caricato il pacchetto `codicefiscaleitaliano.sty` nel preambolo del documento, per controllare il codice fiscale di Maria Angelini, nata ad Urbino il 24 agosto 1908, NGLMRA08M64L500N, basta dare il comando :

`\codicefiscaleitaliano{NGLMRA08M64L500N}`
 anche nella forma:

`\codicefiscaleitaliano`
 `{NGL MRA 08M64 L500 N}`

e viene solo trascritto nel file `.log` che quel codice fiscale è OK; se invece si fosse commesso un qualunque errore di battitura, l'esecuzione del programma di composizione si sarebbe arrestato con un vistoso messaggio d'errore che non può sfuggire all'utente più distratto, in modo che possa provvedere alle correzioni o alle verifiche necessarie. Si noti ancora che se si usa il comando asteriscato il pacchetto, oltre alla verifica, scrive anche il codice nella stessa forma come lo si è immesso nell'argomento del comando; per esempio si ottiene: NGL MRA 08M64 L500 N se si usa il codice scritto come indicato poco sopra.

Bisogna osservare anche che tutta la sintassi dei comandi primitivi è descritta nel T_EXbook, (KNUTH, 1984); siccome però si sono usati dei comandi “più primitivi” del solito, è possibile che anche un utente esperto non ci si ritrovi se non rivedendo con pazienza tutte le informazioni necessarie sul T_EXbook.

Ho scritto questo articoletto solo per mostrare un'altra cosa insolita fra le tante che si possono fare con il sistema T_EX; penso e spero che non si tratti solo di un mio passatempo, ma che la cosa possa essere effettivamente utile, specialmente per quegli utenti italiani del sistema T_EX che scrivono documenti legali, fiscali e simili, dove è necessario scrivere spesso i codici fiscali.

Riferimenti bibliografici

KNUTH, D. E. (1984). *The T_EXbook*. Addison Wesley, Reading, Massachusetts, 18^a edizione.

MINISTRO DELLE FINANZE (1976). «Sistemi di codificazione dei soggetti da iscrivere all'anagrafe tributaria». Supplemento ordinario della Gazzetta Ufficiale n. 345 del 29 dicembre 1976.

▷ Claudio Beccari
 Villarbasse (TO)
 claudio dot beccari at gmail
 dot com

Grafica ad oggetti con LuaTeX

Roberto Giacomelli

Sommario

Nell'articolo si sperimentano le potenzialità espressive e l'efficacia di un *linguaggio ad oggetti* per la grafica in LuaLaTeX cercando di delineare quali vantaggi otterrebbe l'utente programmando col nuovo paradigma rispetto ai tradizionali linguaggi di comandi come PGF, pstricks o METAPOST.

L'introduzione di LuaTeX permette di scrivere codice Lua direttamente in un sorgente TeX e questo rende possibile avvalersi di librerie ad oggetti ampliando ulteriormente le potenzialità del nuovo motore di composizione.

Abstract

In this paper we will make an attempt to measure the final TeX user advantages and efficacy of an object oriented language, the programming paradigm recently added to TeX world by the new typesetting engine LuaTeX.

A case study was developed writing a new graphic object library in Lua, based on the PGF package, carried in a source TeX files with a little ad very simple package.

1 Oggetti per TeX

Se esaminiamo un sorgente LaTeX noteremo una serie di *macro*, ciascuna con un numero variabile di argomenti ed opzioni, che può assumere anche la forma strutturata degli *ambienti* tramite codici di apertura e chiusura. La sintassi di questi comandi è quella prevista dal formato creato da Leslie Lamport e dai vari pacchetti d'estensione nell'idea semplice del paradigma *imperativo*.

Possiamo riassumere queste caratteristiche di LaTeX nelle seguenti affermazioni:

- *Paradigma*: linguaggio macro con comandi ed ambienti;
- *Programmazione utente*: no.

Questo approccio linguistico ha consentito lo sviluppo ininterrotto del sistema TeX dalla fine degli anni settanta ad oggi, rimanendo sostanzialmente invariato nel supportare il notevole numero di nuove e migliorate funzionalità tipografiche a disposizione dell'utente.

Qualche anno fa, quasi per caso giocando con Scite, l'editor programmabile in linguaggio Lua, alcuni sviluppatori proposero di ripetere l'idea includendo l'interprete Lua nel motore di composizione più importante ovvero pdfTeX. Nacque così LuaTeX,

ufficialmente in fase di pre-release, con l'intenzione di colmare le lacune di TeX evidenziate con il confronto con altri linguaggi di programmazione.

Con LuaTeX infatti è possibile scrivere codice Lua direttamente in un sorgente TeX così che possono essere sviluppati pacchetti in modo molto più efficiente e dalle funzionalità più varie, dalla lettura di database server all'accesso al web. Nuovamente aumentano le funzionalità a disposizione, sempre più potenti e generali, ma il modo di utilizzarle si basa ancora su un linguaggio a sequenza di comandi dove la programmazione è nascosta all'utente e celata nelle librerie.

Il progetto dei formati basati su TeX, come LaTeX e ConTeXt, fondato su un linguaggio il più possibile semplice, non cambia con LuaTeX. Secondo questa idea, come per il codice TeX incluso nei pacchetti d'estensione, anche il codice Lua dovrà rimanere nascosto all'utente finale. Tuttavia un linguaggio potente e facile da usare consentirebbe agli utenti di essere più efficienti nel loro lavoro e ciò gioverebbe all'accuratezza dei documenti.

In questo scenario il paradigma della programmazione ad oggetti potrebbe svolgere un ruolo importante nel contribuire a creare strutture linguistiche più intuitive ed anche più creative per l'utente.

Lua rende possibile ciò poiché offre un supporto di base — e per dirla tutta anche assai poco rigoroso — alla programmazione ad oggetti ma sufficiente per le caratteristiche di un linguaggio interpretato che segue una filosofia minimalista e tende ad essere essenziale in tutto.

Il linguaggio del sistema TeX si evolverebbe per divenire essenzialmente una struttura basata su questi due punti:

- *Paradigma*: linguaggio macro e ad oggetti;
- *Programmazione utente*: sì, in Lua con librerie ad oggetti.

1.1 Una grafica ad oggetti

Per sperimentare un linguaggio ad oggetti indirizzato direttamente agli utenti si è scelto di studiare il caso di una libreria grafica, che ben rappresenta da una parte le difficoltà di programmazione degli utenti e dall'altra il diretto rapporto tra enti geometrici ed oggetti.

Nell'articolo verranno illustrate alcune conoscenze di base sia di Lua sia dei concetti del paradigma di programmazione ad oggetti che introduce nei programmi un più elevato livello d'astrazione.

In seguito verrà presentata la libreria sperimentale **GO**L, acronimo di *Graphic Object Library* ed il modo in cui è stata implementata attraverso la grafica ad oggetti, assieme alla sintassi in Lua con cui l'ipotetico utente/programmatore svilupperebbe i propri *disegni ad oggetti*.

2 Programmazione ad oggetti

2.1 Basi della OOP

Dal punto di vista dell'utente, il paradigma della Object Oriented Programming si basa sulla manipolazione di entità indipendenti chiamate *oggetti* attraverso funzioni chiamate *metodi*. Ciascun oggetto è caratterizzato dall'insieme dei suoi metodi che ne definiscono il comportamento, il quale è identico per tutti quelli di una stessa famiglia chiamata *classe*, una sorta di prototipo che rappresenta un "tipo di dati".

Nel sorgente l'oggetto dovrà prima essere creato indicando la classe di appartenenza, e successivamente potrà essere manipolato attraverso i *metodi* che agiranno sullo stato interno del singolo oggetto in lettura o in scrittura, fornendo servizi utili.

La creazione di un oggetto avviene con un metodo particolare chiamato *costruttore* che naturalmente può prevedere dei parametri, e che restituisce un riferimento al nuovo elemento che verrà memorizzato in una variabile con un'istruzione di assegnamento, come nel seguente pseudo-codice:

```
NomeTipo var = new NomeClasse(<args>);
```

Nonostante la variabile sia tecnicamente solo un contenitore, per il programmatore rappresenta invece l'oggetto poiché il nome del riferimento è l'elemento sintattico dopo il quale egli scrive, separandolo con un carattere punto ., il nome dei metodi lanciandone così l'esecuzione.

Questa sintassi, nota comunemente come *dot notation* e rappresentata nella prossima linea di codice, riassume l'essenza del paradigma ad oggetti, nel quale dati ed algoritmi sono assemblati in un'unica struttura semantica.

```
var.metodo(<args>);
```

Le idee di base del paradigma ad oggetti sono concretizzate dai vari linguaggi graduandone le caratteristiche fondamentali: per esempio in Java, linguaggio completamente ad oggetti, il codice per la creazione ed il disegno di un rettangolo grafico in un documento PDF con la libreria iText (LOWAGIE e ALTRI, 2012) potrebbe essere simile al seguente frammento di codice:

```
// creazione del documento
Document doc = new Document();

// creazione di un oggetto grafico contenitore
Graphic g = new Graphic();

// aggiungiamo ad esso un rettangolo
g.rectangle(50,50,200,300);
```

```
// aggiungiamo la figura al documento
doc.add(g);
```

Nel listato è possibile notare che la creazione di nuove variabili comporta l'obbligo di specificarne il *tipo* premettendone il nome alla variabile. Questa importante caratteristica fa di Java un linguaggio a tipizzazione statica, qualità che ne incrementa la robustezza. Nei linguaggi a tipizzazione dinamica come Lua e Python, invece, la determinazione del tipo è rimandata alla fase di esecuzione sacrificando la robustezza dei programmi a favore della semplicità.

Volendo tradurre in Lua il precedente frammento in Java¹ dovremo sostituire il punto della *dot notation* con il carattere : — per il motivo che sarà chiaro in seguito — ed ai doppi backslash // i doppi trattini -- per indicare i commenti di riga. Potremo anche eliminare i simboli di terminazione ; perché facoltativi in Lua e dovremo eliminare qualsiasi dichiarazione di tipo per ottenere il seguente codice:

```
-- creazione del documento
doc = Document:new()
--... eccetera

-- creazione di un oggetto grafico contenitore
g = Graphic:new()

-- aggiungiamo ad esso un rettangolo
g:rectangle(50,50,200,300)

-- aggiungiamo la figura al documento
doc:add(g)
```

La differenza principale tra il codice Java e la sua traduzione in Lua è che in Lua non esiste la parola chiave **new** che istanzia il nuovo oggetto in memoria chiamandone il costruttore. Ed ancora, in Lua il costruttore risulta essere un metodo qualsiasi dell'oggetto — addirittura non obbligatorio — che lo sviluppatore della libreria potrebbe chiamare in qualsiasi altro modo mentre in Java assume il nome della classe.

Il linguaggio Lua non è quindi progettato con gli stessi obiettivi di Java: non possiede un controllo preventivo del tipo — i relativi errori di programmazione emergono solamente nella fase di esecuzione — non prevede una chiara e semplice modalità di generazione dell'oggetto essendo privo della parola chiave **new**, ma sfrutta funzionalità tortuose poco eleganti e non impone una struttura sintattica unica ma lascia al programmatore la libertà e la responsabilità che ne deriva, di sviluppare soluzioni sintattiche alternative.

Tuttavia, Lua offre pieno supporto ai principi del paradigma ad oggetti senza perdere le caratteristiche generali di un linguaggio semplice da usare.

1. Un buon testo per imparare Java è il seguente (CORNELL e HORSTMANN, 1997) anche se l'edizione italiana risale al 2007.

2.2 Campi e metodi

Nella programmazione OOP un oggetto è un'istanza di classe indipendente, ovvero è caratterizzato da un proprio stato interno determinato dal valore assunto dai *campi*, ovvero riferimenti ad altri oggetti o tipi semplici come numeri e stringhe. I *metodi* sono le funzioni *esposte* all'utente, che possono modificare lo stato interno dell'oggetto e che compiono le elaborazioni a cui è demandata la classe stessa.

L'espressione *la classe espone i metodi* significa che essi costituiscono l'interfaccia con cui l'utente interagisce con l'oggetto. Una volta che l'insieme dei metodi è stato fissato, all'esterno non sarà visibile né lo stato interno dell'oggetto, né l'implementazione stessa dei metodi.

Questo significa che il programmatore della classe, nello sviluppo del codice, ha il solo obbligo di mantenere la sintassi prevista dall'interfaccia *pubblica*, nomi e argomenti dei metodi, ma tutta la libertà di modificare le strutture interne *private*, come il codice dei metodi, potendo così migliorare la libreria senza che gli utilizzatori debbano modificare il proprio codice.

I linguaggi OOP prevedono quindi la possibilità di dichiarare come *privati* campi e metodi. Lua non offre invece alcun meccanismo di protezione, semplicemente contando sul comportamento corretto dell'utente della libreria.

2.3 Ereditarietà

Disponendo di una classe che rappresenti un dato concetto, possiamo implementare allo stesso modo altri concetti legati alla realtà del problema, ma ben presto ci renderemo conto che esiste tra loro una relazione. L'*ereditarietà* è il meccanismo che consente di creare una classe da un'altra estendone le funzioni e modificandone il comportamento, per rappresentare una gerarchia di elementi.

Per esempio, in una libreria di disegno potrebbero essere rappresentate le figure del rettangolo, del cerchio, della linea, ecc., ma ciascun oggetto è caratterizzato da proprietà comuni come lo spessore di linea ed un metodo che lo disegna sulla tela, e dal fatto che tutti sono enti geometrici.

La proprietà comune dello spessore di linea può essere rappresentata da un campo numerico in un oggetto geometrico generico, mentre il metodo per disegnare un oggetto reale dovrà essere implementato di volta in volta a seconda del tipo di geometria rappresentata.

Invece di scrivere sempre lo stesso codice per implementare lo spessore di linea in tutte le classi, forma geometrica per forma geometrica, non conviene ereditare questa proprietà *comune a tutte* le figure da un'unica classe?

Questa classe comune è detta classe padre, o classe base, od ancora *superclasse*. Nel caso particolare dell'esempio è anche una classe *astratta*

perché non rappresenta direttamente una forma geometrica.

Tutte le classi geometriche erediteranno le proprietà comuni, definendo il proprio metodo di disegno sempre con lo stesso nome, struttura che sintetizza perfettamente il concetto d'interfaccia.

2.4 Ingegneria del software

Da queste brevi descrizioni del paradigma della programmazione ad oggetti possono essere desunte alcune considerazioni generali riassuntive.

Un oggetto è una sorta di contenitore dove trovano posto sia i dati che le funzioni che li elaborano. Si tratta quindi di un componente software ideato per una più efficiente organizzazione del lavoro di sviluppo dei programmi.

Inoltre, la costruzione ad oggetti introduce la possibilità di creare strutture in grado di rappresentare concettualmente un problema. Si tratta quindi di una metodologia di comprensione e di apprendimento per gradi di sviluppo che consente di costruire su basi solide buone soluzioni software senza che la complessità ne rallenti troppo l'evoluzione.

3 Gli oggetti in Lua

Il linguaggio Lua si fonda sull'essenzialità tanto che supporta la programmazione ad oggetti utilizzando quasi esclusivamente le proprie risorse di base senza introdurre nessun nuovo costrutto. In particolare Lua implementa gli oggetti utilizzando l'unica struttura dati disponibile nel linguaggio, la *tabella*, a cui vengono affiancate particolari funzionalità dette *metatabelle* e *metametodi*.

In questa sezione esamineremo con esempi di codice come viene implementata la programmazione ad oggetti in Lua rimandando al testo di riferimento (IERUSALIMSKY, 2006) per la trattazione di ulteriori argomenti di cui non si farà cenno, come la gestione della memoria tramite *garbage collector* ed alcuni dettagli sulla sintassi per la definizione delle funzioni e dei suoi parametri.

Cominceremo col descrivere la tabella con le sue particolarità e caratteristiche che la fanno assomigliare ad un oggetto tanto da costituire la base per tutta la programmazione OOP in Lua, per passare sempre più nel dettaglio del linguaggio.

Per la comprensione dei meccanismi OOP di Lua è caldamente consigliato ripetere gli esempi proposti, eventualmente utilizzando la modalità interattiva al terminale (è sufficiente digitare alla linea di comando `lua` per entrarvi) o inserendo il codice in file di testo ed eseguendoli sempre al terminale con il comando `lua nomefile`.

3.1 Le tabelle di Lua

In Lua esiste solo una struttura dati complessa con cui si implementa quello che in altri linguaggi viene espresso con molte diverse entità come gli

array, le *liste*, gli *alberi*, eccetera. Questa struttura è chiamata *tabella*, ovvero un array associativo in cui possono essere inseriti valori attraverso una chiave che può essere un numero o una stringa od anche una funzione.

Se le chiavi sono numeri interi rappresenteremo un array, detto anche vettore, ovvero un insieme di valori ciascuno con il proprio indice, se invece i valori sono tabelle rappresenteremo strutture ad albero, peraltro senza limiti di complessità, se ancora sono stringhe rappresenteremo un dizionario.

Il linguaggio prevede che per creare una tabella si debba usare un costruttore; nella sua forma più semplice questo è una coppia di parentesi graffe {}.

Ciascun elemento di una tabella viene restituito specificandone la chiave tra parentesi quadre oppure specificando dopo un carattere punto la chiave come fosse un codice, ma solo se questa è una stringa di caratteri²:

```
local mytab = {}          --> nuova tabella in mytab

mytab["primo"] = 1        --> aggiunta di elementi
mytab.secondo = 2

-- ed in 'dot notation':
print(mytab.primo)        --> stampa 1
-- specificando la chiave
print(mytab["secondo"])   --> stampa 2

-- chiave espressa da una variabile
local seckey = "secondo"
print(mytab[seckey])      --> stampa 2
```

Dall'introduzione anche solo così veloce delle tabelle si può intravedere il carattere del linguaggio Lua: semplice ed essenziale ma anche piuttosto potente ed elegante.

Ironia della sorte, in Lua un particolare tipo di costruttore per le tabelle è stato ideato su ispirazione proprio della sintassi dei record di BibTeX, per cui possiamo istanziare la stessa tabella `mytab` dell'esempio precedente con questa compatta sintassi e stamparne poi i dati con un ciclo `for` che funziona con l'iteratore predefinito `pairs()`:

```
-- costruttore compatto
local mytab = {primo = 1, secondo=2}

-- iterazione su tutti gli elementi
for k, v in pairs(mytab) do
    print(k, v)
end
```

La tabella assume il ruolo di vettore con indici interi a partire da 1 — e non da zero come in molti altri linguaggi — se nel costruttore in forma compatta non si specificano chiavi. Ecco un esempio in cui costruiamo una tabella contenente le prime sette lettere dell'alfabeto per poi stamparle in maiuscolo con un ciclo `for` semplice:

2. La parola chiave `local` utilizzata nelle assegnazioni fa in modo che la variabile venga eliminata dalla memoria quando termina il blocco di codice in cui è creata.

```
-- indici:      1   2   3   4   5   6   7
local mytab = {"a","b","c","d","e","f","g"}

-- ciclo semplice
for i = 1,7 do
    print(string.upper(mytab[i]))
end
```

3.2 Funzioni di prima classe

In Lua le funzioni sono strutture di *prima classe*, possiedono cioè la proprietà di poter essere assegnate a variabili che assumeranno il tipo *function*. Ciò rende possibile memorizzare in una tabella anche funzioni e ciò è particolarmente importante non solo per dotare il linguaggio del supporto alla programmazione funzionale, ma anche per implementare facilmente i metodi di una classe nella programmazione ad oggetti.

Come esempio, creeremo una nuova funzione `print()` semplicemente assegnando alla variabile in sovrascrittura, una nuova funzione che fa uso dell'operatore di concatenazione delle stringhe `..` per la stampa personalizzata:

```
-- tipo di una funzione:
print(type(print)) --> stampa 'function'

oldprint = print    --> ref alla funzione originale

print = function (val) -- sintassi funzione anonima
    oldprint("Hello "..val.."!")
end

print("world") --> stampa 'Hello world!'
print(2012)    --> stampa 'Hello 2012!'
```

Una funzione può essere costruita anche in *sintassi anonima* in cui non compare nessun nome di funzione, costruito utile quando un argomento è una funzione il cui nome è ininfluente. Questo breve esempio chiarisce i due diversi modi di costruzione delle funzioni in Lua:

```
-- Definizioni di funzioni:
-- sintassi canonica
function stampa(val)
    print('-'..val..'-' )
end

-- sintassi anonima della stessa funzione
-- perfettamente equivalente alla precedente
stampa = function (val)
    print('-'..val..'-' )
end
```

In realtà, è il compilatore Lua che rende possibile per l'utente la creazione di funzioni per mezzo della sintassi canonica, nella quale il nome della funzione — che non è altro che il nome della variabile che la conterrà — compare tra la parola chiave `function` e le parentesi tonde che racchiudono eventuali argomenti³.

3. Spesso ci si riferisce a queste funzioni fornite dietro le quinte dal compilatore con l'espressione *zucchero sintattico*.

3.3 Un rettangolo

Avvicinandoci agli obiettivi dell'articolo, costruiremo una classe per rappresentare un rettangolo. Si tratta di un ente geometrico definito da due soli parametri: la larghezza e l'altezza dei lati.

Un primo tentativo potrebbe essere questo:

```
-- prima tentativo di implementazione
-- di una classe rettangolo
Rectangle = {} -- creazione tabella (oggetto)

-- creazione di due campi
Rectangle.width = 12
Rectangle.height = 7

-- un primo metodo assegnato direttamente
-- ad un campo della tabella
function Rectangle.area ()
    -- accesso alla variabile 'Rectangle'
    return Rectangle.larghezza * Rectangle.altezza
end

-- primo test
print(Rectangle.area()) --> stampa 84, OK
print(Rectangle.height) --> stampa 7, OK

-- ancora la prima implementazione
Rectangle = {width = 12, height = 7}

-- un metodo dell'oggetto
function Rectangle.area ()
    -- accesso alla variabile 'Rectangle' attenzione!
    local l = Rectangle.larghezza
    local a = Rectangle.altezza
    return l * a
end

-- secondo test
r = Rectangle -- creiamo un secondo riferimento
Rectangle = nil -- distruggiamo il riferimento originale

print(r.width) --> stampa 12, OK
print(r.area()) --> errore!
```

Il problema sta nel fatto che nel metodo `area()` compare il particolare riferimento alla tabella `Rectangle` che invece deve poter essere qualunque. La soluzione non può che essere l'introduzione del riferimento all'oggetto come parametro esplicito nel metodo stesso, ed è la stessa utilizzata — in modo nascosto ma vedremo che è possibile nascondere il riferimento anche in Lua — dagli altri linguaggi di programmazione che supportano gli oggetti.

Secondo quest'idea dovremo riscrivere il metodo `area()` in questo modo (in Lua il riferimento esplicito all'oggetto *deve* chiamarsi `self` pertanto abituiamoci fin dall'inizio a questa convenzione così da poter generalizzare la validità del codice):

```
-- seconda tentativo
Rettangolo = {larghezza=12, altezza=7}
```

```
-- il metodo diviene indipendente dal particolare
-- riferimento all'oggetto:
function Rettangolo.area ( self )
    return self.larghezza * self.altezza
end

-- ed ora il test
myrect = Rettangolo
Rettangolo = nil -- distruggiamo il riferimento

print(myrect.larghezza) --> stampa 12, OK
print(myrect.area(myrect)) --> stampa 84, OK
-- funziona!
```

Fino ad ora abbiamo costruito l'oggetto sfruttando le caratteristiche della tabella e la particolarità che consente di assegnare una funzione ad una variabile, ma da questo momento entra in scena l'operatore due punti : nella chiamata di funzione. Si tratta di *zucchero sintattico*, ovvero una sorta di aiuto fornito al programmatore dal compilatore, in questo caso per rendere *implicito* il passaggio del riferimento.

L'operatore `:` fa in modo che le seguenti due espressioni siano perfettamente equivalenti, anche se le rende differenti dal punto di vista concettuale agli occhi del programmatore:

```
-- forma classica:
myrect.area(myrect)

-- forma implicita
-- (self prende lo stesso riferimento di myrect)
myrect:area()
```

Questo operatore è il primo nuovo elemento di Lua introdotto per supportare la programmazione orientata agli oggetti. Se si accede ad un metodo memorizzato in una tabella con l'operatore due punti : anziché con l'operatore `.`, l'interprete Lua aggiungerà *implicitamente* un primo parametro con il riferimento alla tabella stessa a cui assegnerà il nome di `self`.

3.4 Metatabelle e metametodi

Il salto definitivo nella programmazione OOP consiste nel poter costruire una *classe* senza ogni volta assemblare i campi ed i metodi ma introducendo un qualcosa che faccia da stampo per gli oggetti.

Abbiamo fatto cenno a Java in cui esiste la parola chiave `class` per la definizione di nuove classi, ma in Lua, dove la semplicità è sempre importante e dove devono continuare ad essere supportati più stili di programmazione, non esistono né nuove parole chiave né una sintassi specifica.

In Lua l'unico meccanismo disponibile per compiere questo ultimo importante passo consiste nelle *metatabelle*, normali tabelle contenenti funzioni dai nomi prestabiliti che vengono chiamate quando si verificano particolari eventi come l'esecuzione di un'espressione di somma tra due tabelle con l'operatore `+`. Ogni tabella può essere associata ad una metatabella e questo consente di creare degli insiemi di tabelle che condividono una stessa aritmetica.

I nomi di queste funzioni particolari dette *metamethodi* iniziano tutti con un doppio trattino basso, per esempio nel caso della somma sarà richiesta la funzione `__add()` della metatabella associata alle due tabelle addendo, e se non esiste verrà generato un errore.

Il metametodo più semplice di tutti è chiamato `__tostring()` e viene invocato se una tabella è data come argomento alla funzione `print()` per ottenere il valore effettivo da stampare: se ovvio è memorizzato nella metatabella associata altrimenti verrà stampato l'indirizzo di memoria della variabile:

```
-- un numero complesso
complex = {real = 4, imag=-9}
print(complex) --> stampa: 'table: 0x9eb65a8'

-- un qualcosa di piu utile: metatabella in sintassi
-- anonima con il metametodo __index
mt = {}
mt.__tostring = function (c)
    local r = string.format("%.2f",c.real)
    if c.imag == 0 then -- numero reale
        return "(.r.)"
    end
    -- numero complesso
    local i = string.format("%.2f",c.imag)
    return "(.r..i.)"
end

-- assegnazione della metatabella mt a complex
setmetatable(complex, mt)

-- riprovo la stampa
print(complex) --> stampa '(4.00,-9.00)'
```

3.5 Il metametodo `__index()`

Il metametodo che interessa la programmazione ad oggetti in Lua è `__index()`. Esso interviene quando viene chiamato un campo di una tabella che non esiste e che normalmente restituirebbe il valore `nil`, il tipo nullo in Lua. Un esempio di codice chiarirà il meccanismo:

```
-- una tabella con un campo 'a'
-- ma senza un campo 'b'
t = {a = 'Campo A'}

print(t.a) --> stampa 'Campo A'
print(t.b) --> stampa 'nil'

-- la metatabella e metametodo
mt = {}
mt.__index = function ()
    return 'Attenzione: campo inesistente!'
end

-- assegniamo 'mt' come metatabella di 't'
setmetatable(t, mt)

-- adesso riproviamo ad accedere al metodo b
print(t.b) --> stampa 'Attenzione: campo inesistente!'
```

Tornando all'oggetto `Rettangolo`, riscriviamo il codice creando adesso una tabella che assumerà il ruolo concettuale di una vera e propria *classe* definendo i campi ed i metodi:

```
-- una nuova classe Rettangolo (campi):
Rettangolo = {larghezza=10, altezza=10}

-- un metodo:
function Rettangolo:area()
    return self.larghezza * self.altezza
end

-- creazione metametodo
Rettangolo.__index = Rettangolo

-- un nuovo oggetto Rettangolo
r = {}
setmetatable(r, Rettangolo)

print( r.larghezza ) --> stampa 10, Ok
print( r:area() )    --> stampa 100, Ok
```

Queste poche righe di codice racchiudono il meccanismo un po' tortuoso della creazione di una nuova classe in Lua: abbiamo infatti assegnato ad una nuova tabella `r` la metatabella con funzione di classe `Rettangolo`. Quando viene richiesta la stampa del campo `larghezza`, poiché tale campo non esiste nella tabella vuota `r`, viene ricercato il metametodo `__index()` nella metatabella associata che è appunto la tabella `Rettangolo`.

A questo punto, il metametodo restituisce semplicemente la tabella `Rettangolo` stessa e questo fa sì che divengano disponibili tutti i campi ed i metodi in essa contenuti. Il campo `larghezza` ed il metodo `area()` del nuovo oggetto `r` sono in realtà quelli definiti nella tabella `Rettangolo`.

Se volessimo creare invece un rettangolo assegnando direttamente la dimensione dei lati, dovremmo semplicemente crearli con i nomi previsti dalla classe: `larghezza` e `lunghezza`. Il metodo `area()` sarà ancora caricato dalla tabella `Rettangolo` ma i campi numerici con le nuove misure dei lati saranno quelli interni dell'oggetto `r` poiché semplicemente esistono e perciò non sarà presa in causa la metatabella. Questa costruzione funziona perfettamente ma lascia al programmatore una situazione di scomodità che deriva dal progetto stesso di Lua e che può essere in parte sanata con l'introduzione del costruttore, come vedremo meglio a fine sezione.

Le chiamate alle metatabelle ed ai metametodi complicano la comprensione del funzionamento del meccanismo di stampa, compito della classe. Può sembrare che ciò influisca negativamente nella scrittura di programmi ad oggetti in Lua. Tuttavia quello che è importante è comprendere il meccanismo delle metatabelle svolto dietro le quinte, poiché in fase di sviluppo il linguaggio ci apparirà concettualmente simile ad una classe.

Le cose da ricordare di scrivere nel codice sono l'impostazione del metametodo `__index()` con la tabella di classe e l'esecuzione della funzione `setmetatable()` per impostare la tabella dell'oggetto stessa come metatabella.

Riproponendo ancora il problema di rappresentare il concetto di rettangolo, completiamo il quadro introducendo quello che adesso ci appare come un

normale metodo ma che concettualmente assumerà il ruolo di costruttore della classe che chiameremo *sempre* con il nome convenzionale `new()`. Il lavoro che dovrà svolgere sarà quello di inizializzare i campi argomento in una delle tante modalità possibili una volta effettuato l'eventuale controllo di validità degli argomenti.

Il codice completo della classe `Rettangolo` è il seguente:

```
-- nuova classe Rettangolo (campi con valore di default)
Rettangolo = {larghezza = 1, altezza = 1}

-- metametodo
Rettangolo.__index = Rettangolo

-- metodo di classe
function Rettangolo:area()
    return self.larghezza * self.altezza
end

-- costruttore di classe
function Rettangolo:new( o )
    -- creazione nuova tabella
    -- se non ne viene fornita una
    o = o or {}

    -- assegnazione metatabella
    setmetatable(o, self)

    -- restituzione riferimento oggetto
    return o
end

-- codice utente -----
r = Rettangolo:new{larghezza=12,altezza=2}

print(r.larghezza) --> stampa 12, Ok
print(r:area())    --> stampa 24, Ok

q = Rettangolo:new{larghezza=12}
print(q:area())    --> stampa 12, Ok
```

Il costruttore accetta una tabella come argomento, altrimenti ne crea una vuota e la restituisce non appena le ha assegnato la metatabella. In questo modo una tabella qualsiasi entra a far parte della famiglia `Rettangolo`. Il costruttore passa al metodo `new()` il riferimento implicito a `Rettangolo` grazie all'uso dell'operatore `:`, per cui `self` punterà a `Rettangolo`.

Quando nel codice viene passata una tabella con uno o due campi sulle misure dei lati al costruttore, l'oggetto disporrà delle misure come valori interni effettivi, cioè dei parametri indipendenti che costituiscono il suo stato interno. Lo sviluppatore può fare anche una diversa scelta, quella di considerare la tabella argomento del costruttore come semplice struttura di chiavi/valori da sottoporre al controllo di validità e poi includere in una nuova tabella con modalità e nomi che riguardano solo l'implementazione interna della classe.

3.6 Questa volta un cerchio

Per capire ancor meglio i dettagli e renderci conto di come funziona il meccanismo nascosto ed auto-

matico delle metatabelle, costruiamo un oggetto `Cerchio` che annoveri fra i suoi metodi uno che modifichi il valore del raggio aggiungendo una misura specificata:

```
Cerchio = {radius=0}
Cerchio.__index = Cerchio

function Cerchio:area()
    return math.pi*self.radius^2
end

function Cerchio:addToRadius(v)
    self.radius = self.radius + v
end

function Cerchio:__tostring()
    local frmt = 'Sono un cerchio di raggio %0.2f.'
    return string.format(frmt, self.radius)
end

-- il costruttore attende l'eventuale valore del raggio
function Cerchio:new(r)
    local o = {}

    if r then
        o.radius = r
    end

    setmetatable(o, self)
    return o
end

-- codice utente -----
o = Cerchio:new()
print(o) --> stampa 'Sono un cerchio di raggio 0.00'

o:addToRadius(12.342)

print(o) --> stampa 'Sono un cerchio di raggio 12.34'
print(o:area()) --> stampa '478.54298786.'
```

Nella sezione del codice utente viene dapprima creato un cerchio senza fornire alcun valore per il raggio. Ciò significa che quando stampiamo il valore del raggio con la successiva istruzione otteniamo 0, il valore di default del raggio dell'oggetto `Cerchio`, per effetto della chiamata ad `__index()` della metatabella.

Fino a questo momento la tabella dell'oggetto `o` non contiene alcun campo `radius`. Cosa succede allora quando viene lanciato il comando `o:addToRadius(12.342)`?

Il metodo `addToRadius()` contiene una sola espressione. Come da regola viene prima valutata la parte a destra ovvero `self.radius + v`. Il primo termine assume il valore previsto in `Cerchio` — quindi zero — grazie al metametodo, e successivamente il risultato della somma uguale all'argomento `v` è memorizzato nel campo `o.radius` che viene creato effettivamente solo in quel momento ed eventualmente utilizzato successivamente in lettura o scrittura.

4 L'idea sulla carta

Dopo questa lunga introduzione al mondo della programmazione ad oggetti possiamo addentrarci nella messa a punto della libreria `GOL`, scelta come campo di studio e sperimentazione con l'obiettivo di consentire all'utente finale di scrivere codice Lua ad oggetti all'interno di un sorgente `LuaLATEX`.

4.1 Struttura del codice

La strategia per l'implementazione dell'idea di base è questa: durante l'esecuzione il codice Lua rilascia, nel flusso di token per `TEX`, opportune macro `TikZ` corrispondenti alle istruzioni di disegno. Terminata l'esecuzione del codice Lua, il controllo torna a `TEX` che elabora le macro grafiche esattamente come se le avesse digitate esplicitamente l'utente, producendo effettivamente il disegno.

I passi da fare per generare un disegno sono i seguenti:

- Aprire un ambiente `tikzpicture`;
- scrivere il codice Lua che rilasci i token dei comandi `TikZ`;
- chiudere l'ambiente `tikzpicture`;

Considerata la disponibilità in `LuaTEX` della libreria grafica `mplib`, avremmo potuto più coerentemente scegliere `METAPOST` per la produzione effettiva del disegno. In questo modo senza la necessità di pacchetti esterni, avremmo potuto gestire la generazione della grafica vettoriale interamente in Lua, beneficiando di migliori prestazioni sia in termini di esecuzione sia in termini di memoria.

Avremmo potuto addirittura creare direttamente le primitive grafiche nel documento di uscita senza avvelerci di alcun pacchetto particolare. La scelta di utilizzare `TikZ` come *motore* grafico sottostante può quindi sembrare arbitraria ma rende più semplice la scrittura del codice e crea inoltre un ambiente di sviluppo più adatto alla sperimentazione.

4.2 Il pacchetto `GOL`

Le operazioni elencate precedentemente possono essere effettuate direttamente dall'utente, tuttavia risulta più coerente mettere a disposizione un ambiente `gol` all'interno di un pacchetto d'estensione.

Perché all'interno dell'ambiente `gol` si possa scrivere codice Lua ci si può avvalere del pacchetto `luacode`. Questo è in grado di gestire opportunamente i codici di categoria dei token, problema originato dai diversi significati che alcuni simboli assumono in `TEX` ed in Lua. Per esempio, il simbolo percentuale o i backslash (ulteriori informazioni possono essere reperite nella documentazione di questo pacchetto (PÈGOURIÈ-GONNARD, 2012)).

Dopo il caricamento dei pacchetti di base il codice prevede quello della libreria `GOL`, per mezzo

della funzione `require()`. Questo tipo di procedura consente di posizionare la libreria in file separati dove è possibile distribuire opportunamente il codice Lua ed evitare i problemi di interpretazione dei simboli, senza comunque compromettere lo sviluppo di ulteriori librerie basate su `GOL` — magari programmate da contributori esterni — da caricare solo su richiesta dell'utente.

Per ultimo si definisce il nuovo ambiente `gol` con le macro in stile `TEX` di apertura e chiusura degli ambienti componenti, `tikzpicture` e `luacode`, come mostra il seguente listato completo:

```
% lualatex package
\ProvidesPackage{gol}[2012/08/01 v0.1 Graphic
  Object Library]
\RequirePackage{luacode}
\RequirePackage{tikz}

\directlua{require "gol"}
\newenvironment{gol}
{
  \tikzpicture
  \luacode
  {\endluacode
  \endtikzpicture}
\endinput
```

Un primo sorgente di controllo del pacchetto `gol` potrebbe essere il seguente, utile oltre che per verificare che tutto funzioni, anche per comprendere operativamente l'implementazione dell'idea di rilascio di token `TEX` da Lua (occorre compilare il sorgente con `LuaLATEX`):

```
% direttive per l'editor TeX works:
% !TEX program = lualatex
% !TEX encoding = UTF-8 Unicode

\documentclass{article}
\usepackage{fontspec}
\usepackage[italian]{babel}
\usepackage{gol}

\begin{document}
\begin{gol}
-- funzione di disegno
-- x,y -> angolo in basso a sinistra
-- b,h -> dimensioni lati base ed altezza
local function rdraw(x,y, b,h)
  local frmt="\draw (%d,%d) rectangle (%d,%d);"
  tex.sprint(string.format(frmt, x, y, x+b, y+h))
end

-- disegno un quadrato
rdraw(0,0, 2,2)

-- disegno un rettangolo
rdraw(3,0, 2,4)
\end{gol}
\end{document}
```

Nel listato compilabile è definita una funzione chiamata `rdraw()` che *rilascia* i token per `TEX` utilizzando la funzione di `LuaTEX` `tex.sprint()`. Il suo scopo è disegnare rettangoli con una particolare logica sulla definizione della geometria: i primi due parametri sono le coordinate assolute del vertice in basso a sinistra del rettangolo e gli ultimi due sono le dimensioni dei lati.

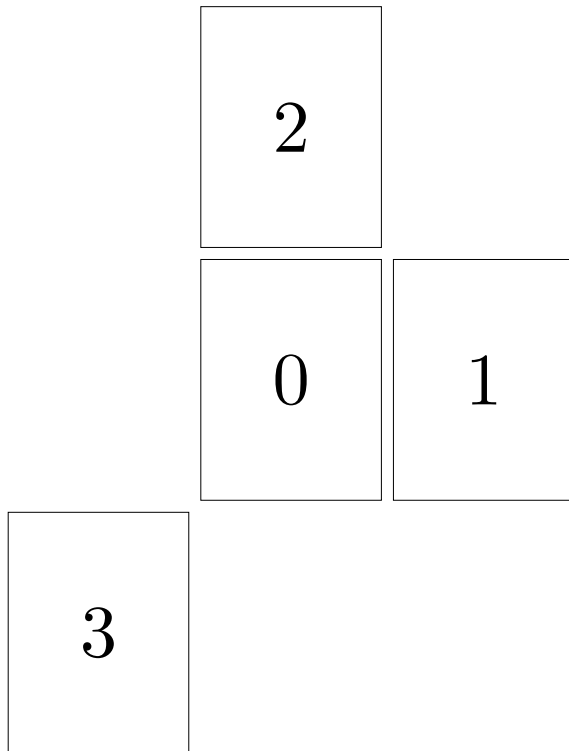


FIGURA 1: Disegno ottenuto con il sorgente Lua $\text{\TeX}$ riportato nel testo, che fa uso di una prima implementazione della libreria GOL a cui è stato aggiunto una numerazione dei rettangoli per facilitare la lettura del codice (immagine scalata rispetto alle dimensioni originali).

Il modello di sorgente Lua $\text{\TeX}$ per l'utilizzo della libreria GOL è il seguente:

```
\documentclass{<classname>}
\usepackage{fontspec}

...{altri pacchetti}
\usepackage{gol}
...{altri pacchetti}

\begin{document}
...
\begin{gol}
...{codice Lua}
\end{gol}
...
\end{document}
```

L'ambiente gol è molto semplice. Non fa nulla se non predisporre la programmazione Lua da parte dell'utente $\text{\TeX}$. Il compito di rendere semplice ed efficiente questa attività è l'obiettivo principale della libreria GOL.

4.3 Primo oggetto

Il primo esempio di costruzione grafica è riportato nel seguente listato, il cui risultato è riportato nella figura 1. In esso viene costruito un rettangolo di larghezza 3 ed altezza 4, che viene poi disegnato attraverso il metodo `draw()` dell'oggetto:

```
% !TEX program = lualatex
% !TEX encoding = UTF-8 Unicode
```

```
\documentclass{standalone}
\usepackage{fontspec}
\usepackage{gol}

\begin{document}
\begin{gol}
r = Rectangle:new{x=3,y=4}

-- disegno
r:draw()           --> 0
r:draw{dx=3.2}     --> 1
r:draw{dy=4.2}     --> 2
r:draw{dx=-3.2,dy=-4.2} --> 3
\end{gol}
\end{document}
```

Già questa prima semplice implementazione introduce un concetto nuovo rispetto al linguaggio tradizionale e già avanzato del pacchetto TikZ: l'oggetto geometrico deve prima essere creato e solo successivamente può essere disegnato. Questa separazione tra ente geometrico e sua rappresentazione grafica caratterizza il modo in cui l'utente manipola gli oggetti ed il disegno rendendo il processo di costruzione del codice più intuitivo e flessibile.

La sintassi di Lua prevede che se si passa ad una funzione un unico argomento e questo è una stringa o una tabella allora è possibile omettere le parentesi tonde. Per questo nel sorgente di esempio d'uso, essendo il costruttore una normale funzione, può sembrare che si sia fatto uso delle parentesi graffe al posto di quelle tonde. In realtà stiamo passando un'unica tabella nella forma compatta del costruttore `{chiave1=valore1, chiave2=valore2, ...}`.

Il codice Lua che implementa il primo oggetto `Rettangolo` prevede la creazione iniziale di una serie di parametri di identificazione come data e numero di versione, la definizione di un semplice costruttore che crea i due campi *geometrici* relativi alle dimensioni dei lati del rettangolo e del metodo `draw()` che accetta come argomento una tabella opzionale con i campi `dx` e `dy` col significato di vettore di spostamento (δ_x, δ_y) .

Il metodo di disegno `draw()` può fornire numerose opzioni relative sia a proprietà grafiche come colore, spessore, tipo linea e riempimento, sia a proprietà geometriche di scalatura, spostamento rispetto al sistema di riferimento locale, e rotazione. Dal lato implementativo non occorre nessuno sforzo di programmazione poiché la gestione di opzioni $\langle chiave \rangle = \langle valore \rangle$ è insita nella tabella di Lua e nel suo costruttore in forma compatta.

Il codice seguente, prima versione della libreria GOL, dovrà trovarsi nel file di nome `gol.lua` in una delle cartelle di sistema a cui $\text{\TeX}$ può accedere:

```
-- prima implementazione dell'oggetto rettangolo
Rectangle = {}           -- init
Rectangle.__index = Rectangle -- metamethodo
Rectangle.classname = 'Rectangle'
Rectangle.version = '0.01' -- version number
Rectangle.date = '20120530' -- library date
```

```

-- costruttore
function Rectangle:new(t)
    local o = {}
    o.x = t.x or 1
    o.y = t.y or 1

    setmetatable(o, self) -- metatabella
    return o
end

-- metodo di disegno
-- v -> vettore di spostamento
function Rectangle:draw(v)
    local dx, dy
    if v then
        dx, dy = v.dx or 0, v.dy or 0
    else
        dx, dy = 0, 0
    end

    -- geometria
    local x1 = -self.x/2 + dx
    local y1 = -self.y/2 + dy
    local x2 = self.x/2 + dx
    local y2 = self.y/2 + dy

    local frmt = '\\draw (%0.3f,%0.3f) rectangle
        (%0.3f,%0.3f);'
    tex.sprint(string.format(frmt, x1,y1,x2,y2))
end

return Rectangle

```

4.4 Griglie

Il metodo `draw()` degli oggetti grafici si presta bene per assumere funzionalità sofisticate grazie alla sintassi di specifica parametri chiave/valore supportata direttamente da Lua, anche con sottotabelle.

Si sono già incontrate le chiavi `dx` e `dy` per disegnare l'oggetto con uno spostamento nel piano ed è naturale implementare nel metodo di disegno ulteriori modalità di generazione anziché creare nuovi metodi. Per esempio, possiamo pensare di aggiungere un parametro `grid` che stampi l'oggetto su una griglia.

Questo nuovo parametro `grid` potrebbe essere una tabella contenente il numero delle righe e delle colonne, la spaziatura orizzontale e verticale e l'angolo di distorsione come nel seguente esempio il cui risultato è riportato in figura 2:

```

\begin{gol}
r = Rectangle:new{x=.75,y=.75}

-- disegno su griglia
r:draw{grid={rows=3,cols=5,dx=1,dy=2,angle=75}}
\end{gol}

```

Il metodo `draw()` assumerebbe un implementazione simile alla seguente:

```

-- metodo di disegno
-- grid -> tabella con specifiche di griglia
function Rectangle:draw(v)
    local dx, dy
    local nx, ny, sx, sy, a
    if v then
        dx, dy = v.dx or 0, v.dy or 0

```

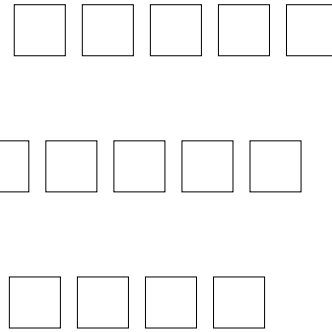


FIGURA 2: Un esempio delle potenzialità di linguaggio del metodo `draw()` di un oggetto grafico, per la stampa su una griglia parametrica (scalato rispetto alle dimensioni originali).

```

    if v.grid then
        ny = v.grid.rows or 1
        nx = v.grid.cols or 1
        sx = v.grid.dx or 1
        sy = v.grid.dy or 1
        a = v.grid.angle or 90
    else
        nx, ny = 1, 1
        sx, sy = 1, 1
        a = 90
    end

    end
else
    dx, dy = 0, 0
    nx, ny = 1, 1
    sx, sy = 1, 1
    a = 90
end

end

-- geometria
local x1 = -self.x/2 + dx
local y1 = -self.y/2 + dy
local x2 = self.x/2 + dx
local y2 = self.y/2 + dy

local frmt = '\\draw (%0.3f,%0.3f) rectangle
    (%0.3f,%0.3f);'
for i=0,nx-1 do
    for j=0,ny-1 do
        local vx = i*sx + j*sy/math.tan(math.pi*a
            /180)
        local vy = j*sy
        tex.sprint(string.format(frmt,
            x1+vx,y1+vy,x2+vx,y2+vy))
    end
end
end
end

```

L'opzione `grid` può anche contenere *funzioni* — in virtù del fatto che in Lua le funzioni sono variabili di prima classe — per personalizzare nodo per nodo il disegno in griglia dell'oggetto. Per esempio, è possibile prevedere un campo `selection` a cui associare una funzione che restituisce un valore *vero/falso* secondo i valori del numero di riga e del numero di colonna della griglia. Di essa potremo facilmente disegnare solo i nodi delle diagonali, oppure realizzare uno schema a scacchiera.

L'implementazione del metodo `draw()` dovrebbe semplicemente interrogare nodo per nodo la funzione `selection` prima di effettuare il disegno

dell'oggetto e, nel caso in cui essa non è specificata dall'utente, creare una funzione sostitutiva che restituisce sempre il valore *vero*:

```
-- metodo di disegno
-- grid -> tabella con specifiche di griglia
function Rectangle:draw(v)
...
... codice iniziale come sopra

-- check sulla funzione 'selection'
local isSel
if v then
  if v.grid then
    isSel = v.grid.selection or
            function () return true end
  end
else
  isSel = function () return true end
end

local frmt = '\\draw (%0.3f,%0.3f) rectangle
              (%0.3f,%0.3f);'
for i=0,nx-1 do
  for j=0,ny-1 do
    if isSel(j+1,i+1) then
      local vx = i*sx +
                j*sy/math.tan(math.pi*a/180)
      local vy = j*sy
      tex.sprint(string.format(frmt,
                               x1+vx,y1+vy,x2+vx,y2+vy))
    end
  end
end
end
```

Per costruire una scacchiera — un buon esempio di applicazione della funzionalità di selezione del disegno nodo per nodo — potremo scrivere un codice simile al seguente:

```
\begin{gol}
local function scacchiera(r,c)
  local s = (r+c)/2 - math.floor((r+c)/2)
  if s==0 then return true else return false end
end

q = Rectangle:new{}

-- disegno con griglia
q:draw{grid={rows=8,cols=8,
              dx=1.1,dy=1.1,
              selection = scacchiera,
            }
}
\end{gol}
```

L'idea si potrebbe ampliare pensando ad un ulteriore campo funzione dell'opzione **grid**, che modifichi le proprietà dell'oggetto prima del disegno nodo per nodo, in funzione della posizione sulla griglia.

4.5 RefPoint e RefAxis

Uno dei vantaggi riconosciuti del disegno programmato è la facilità con cui gli elementi grafici possono essere affinati modificando dimensioni e proprietà procedendo per passi fino al risultato finale, o per adattare una precedente figura di cui disponiamo già del codice. Nella libreria **GOL**, oltre a

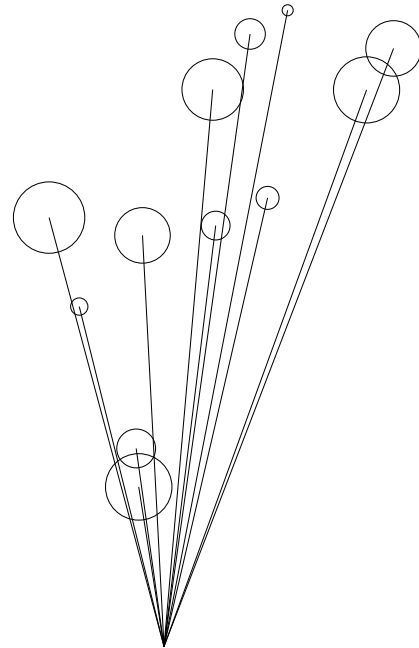


FIGURA 3: Il disegno elementare di un mazzo di fiori formati da un cerchio e da una linea come gambo, di angolo, lunghezza e dimensione casuale, ottenuto con l'utilizzo di un oggetto **RefPoint** studiato per facilitare la costruzione logica e strutturata dei disegni.

questa essenziale caratteristica, si tenta di semplificare anche la costruzione del disegno relazionando un punto od un asse alla geometria dell'oggetto piuttosto che a coordinate fornite esplicitamente.

Introdurremo cioè *oggetti riferimento*, la cui geometria nel piano dipenderà da una particolare caratteristica di un altro oggetto grafico. In particolare esploreremo la classe **RefPoint**, che rappresenterà un punto, e la classe **RefAxis** che rappresenterà un asse.

In questo modo possiamo pensare logicamente al disegno creando dapprima gli elementi principali e successivamente altri elementi ad essi geometricamente vincolati. Per esempio, potremo rappresentare un *fiore* in modo elementare con un cerchio e con una linea che unisce un punto con il centro del cerchio. Una geometria simile si fonda sulla seguente relazione geometrica: la linea del *gambo* ha un estremo coincidente con il centro del cerchio.

Allo scopo di disegnare un quanto mai rudimentale mazzo di fiori, in un ambiente **gol** potremo scrivere il seguente codice il cui risultato è riportato in figura 3:

```
\begin{gol}
-- disegno di una dozzina di fiori casuale
fiore = Circle:new{r=1}
-- creo l'oggetto 'RefPoint'
centroFiore = fiore:getPoint{'center'}

gambo = Line:new{{0,0}, centroFiore}

-- parametri del mazzo
maxangle = 24
maxdim = 8
```

```

for i=1,12 do
  local alfa = math.random(-maxangle,maxangle)
  local dist = maxdim*math.random()+2

  fiore:setxy{polar={90-alfa, dist}}

  fiore:draw{scale=0.5*math.random()}
  gambo:draw()
end
\end{gol}

```

Nel codice, la relazione geometrica gambo/fiore viene espressa nella definizione dell'oggetto `centroFiore` attraverso il metodo `getPoint()` definito nella classe `Cerchio`. Successivamente viene creata una nuova linea con un vertice nell'origine del sistema di riferimento e con l'altro nel centro del cerchio. Modificando la geometria del cerchio non occorre così riaggiornare di conseguenza le coordinate del secondo estremo della linea. All'interno del ciclo `for`, per disegnare il gambo del fiore è sufficiente chiamare il metodo `draw()` dell'oggetto che lo rappresenta.

Tecnicamente, il metodo `getPoint()` restituisce un oggetto `RefPoint` che, secondo la terminologia della programmazione ad oggetti, dovremo chiamare *metodo factory*, poiché in realtà la classe `RefPoint` non possiede alcun costruttore. Questa è conseguenza diretta della relazione geometrica creata dall'utente: solamente l'oggetto `fiore` conosce la posizione nel piano del proprio centro e può così creare il corrispondente oggetto `RefPoint`.

La struttura del codice Lua che implementa la funzionalità, si basa su semplici funzioni memorizzate all'interno di una tabella che dato il raggio del cerchio, restituiscono le coordinate del punto indicato da una chiave, che può essere per esempio 'n' per il punto a nord e 'center' per il centro:

```

Circle.vertexKey = {
  center = function (r) return 0, 0 end,
  n       = function (r) return 0, r end,
  s       = function (r) return 0,-r end,
  e       = function (r) return r, 0 end,
  w       = function (r) return -r, 0 end,
}

```

Nel metodo vero e proprio, si memorizza in un nuovo campo tabella chiamato `refpoints`, sulla base di un indice progressivo ed in sintassi anonima, la funzione che restituisce le coordinate effettive del punto di riferimento.

Eliminando dal codice i controlli di validità degli argomenti per favorirne la comprensione, l'implementazione del metodo `getPoint()` nella classe `Circle` è la seguente:

```

function Circle:getPoint(t)
  local trp = self.refpoints
  local id = #trp + 1
  local fxy = Circle.vertexKey[t[1]]
  trp[id] = function ()
    local x, y = fxy(self.radius)
    return x+self.x, y+self.y
  end
end

```

```

-- RefPoint factory
local rp = {}
rp.object = self
rp.id = id

setmetatable(rp, RefPoint)
return rp
end

```

L'oggetto `RefPoint` necessita di un campo tabella in cui memorizzare l'oggetto riferito ed un campo numerico dove memorizzare l'indice del punto stesso. In questo modo, possono essere creati un numero qualsiasi di punti riferimento per lo stesso oggetto.

Quando il metodo `draw()` della classe `Line` necessita delle coordinate del punto dei suoi estremi, chiama il metodo `getxy()` di ciascuno di essi. Se si tratta di un `RefPoint` il codice recupererà la funzione di calcolo delle coordinate dall'oggetto riferito che una volta eseguita restituirà le corrette coordinate, per esempio con il seguente codice:

```

function RefPoint:getxy()
  local o = self.object
  local id = self.id
  local fxy = o.refpoints[id]
  return fxy()
end

```

Come è possibile osservare, l'oggetto `RefPoint` non sa niente né di che tipo di oggetto si tratti, né dove sia effettivamente il punto riferito. Per ottenere questa generalità è necessario tuttavia che tutti gli oggetti grafici abbiano un campo `refpoints` contenente le funzioni di calcolo delle coordinate. In termini di paradigma, ciò significa che gli oggetti grafici devono implementare una stessa *interfaccia*.

Anche la classe `Line` — più correttamente si tratterebbe della rappresentazione di una polilinea e non del più semplice concetto di segmento supposto nel codice — necessita di un campo tabella chiamato `nodes` in cui salvare nell'ordine gli oggetti punto. Ecco una versione essenziale del metodo `draw()`:

```

function Line:draw(v)
  local dx, dy
  if v then
    dx, dy = v.dx or 0, v.dy or 0
  else
    dx, dy = 0, 0
  end
  -- geometria
  local p = self.nodes
  local p1 = p[1]
  local p2 = p[2]
  local x1, y1 = p1:getxy()
  local x2, y2 = p2:getxy()
  x1, y1 = x1 + dx, y1 + dy
  x2, y2 = x2 + dx, y2 + dy

  local frmt =
    '\\draw (%0.3f,%0.3f) -- (%0.3f,%0.3f);'
  tex.sprint(string.format(frmt, x1,y1,x2,y2))
end

```

Interessante è la *flessibilità* del costruttore dell'oggetto **Line** che nel caso del gambo del fiore, deve gestire elementi diversi: una tabella con due numeri come primo argomento, ed un oggetto **Refpoint** come secondo argomento. Il costruttore può essere qualcosa di simile al seguente codice:

```
function Line:new(t)
  local l = {}
  l.nodes = {}

  for i=1,#t do
    local p = t[i]
    if p.classname then
      l.nodes[#l.nodes+1] = p
    else
      l.nodes[#l.nodes+1] = Point:new{x=p[1],
                                     y=p[2]}
    end
  end
  setmetatable(l, self)
  return l
end
```

Il campo `classname` memorizza il nome della classe ed è qui usato per discriminare tra un punto dato come semplice tabella $\{\langle xcoord \rangle, \langle ycoord \rangle\}$, ed un oggetto **RefPoint** o **Point** (di cui per brevità non se ne riporta il codice).

Altra particolarità che riguarda l'utente, è che è necessario modificare la posizione dell'oggetto **fiore** perché poi il punto riferimento **centroFiore** possa ottenere correttamente la posizione del centro. Avremo potuto disegnare i cerchi nella posizione generata in modo casuale, sfruttando le opzioni `dx` e `dy` del metodo `draw()` ma l'oggetto **RefPoint** avrebbe puntato sempre nell'origine.

4.6 Assemblaggio di un disegno

Un disegno è un insieme di elementi come linee, tratti curvi, testo, e figure geometriche. Quello che la tecnologia GOL vuole favorire è un processo di costruzione per gruppi più semplici per poi procedere al loro assemblaggio.

Potremo quindi sviluppare una nuova classe **Drawing** che aggrega semplicemente oggetti grafici ed il cui metodo `draw()` chiami uno alla volta i metodi `draw()` degli oggetti dell'insieme.

Tuttavia a ben vedere abbiamo introdotto finora oggetti grafici per rappresentare non disegni bensì elementi geometrici. Una collezione di questi tipi non è un disegno ma piuttosto un insieme di entità geometriche anche correlate tra loro. Sulla base di questo ragionamento, la classe con cui rappresentare il concetto di gruppo di entità è un assemblaggio geometrico e pertanto dovrà fornire metodi per la gestione di questo insieme.

Assembly potrebbe essere il nome di questa classe che conterrebbe due tipi diversi di informazioni: la collezione degli oggetti geometrici e, per ciascuno di essi, la definizione di una o più posizioni nel piano. Stiamo quindi pensando non più ad un disegno, ma ad un contenitore in cui possono essere inseriti ed eliminati oggetti geometrici, con un

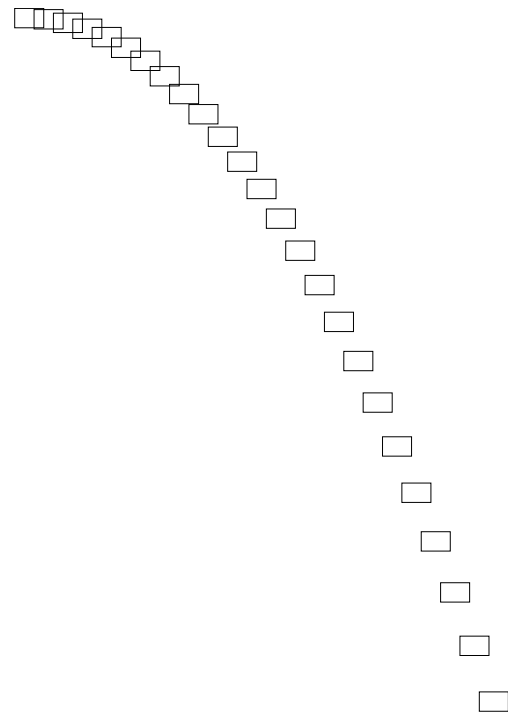


FIGURA 4: Disegno dimostrativo ottenuto utilizzando un oggetto **Assembly** contenente un rettangolo posizionato più volte a simulare il libero spostamento nel campo gravitazionale terrestre (scalato rispetto alle dimensioni originali).

effetto simile a quello dell'installazione e rimozione di un componente software in un sistema.

L'idea è quindi questa: disporre di un contenitore in cui caricare oggetti — che nulla vieta siano essi stessi gruppi di elementi — per poi disporlo più volte sul piano, specificando i parametri di posizione, scala, e rotazione. Alle due tipologie di informazione che definiscono un oggetto **Assembly**, ovvero *l'insieme di oggetti* e la loro *distribuzione nel piano*, corrispondono due azioni riguardanti gli elementi del gruppo: *l'installazione* ed il *dispiegamento*.

L'esempio scelto per sperimentare e comprendere operativamente il concetto ideato per rappresentare l'assemblaggio di più oggetti grafici è mostrato in figura 4 dove un unico rettangolo è posto in diversi punti del piano come se fosse stato lanciato nel campo gravitazionale terrestre con una velocità iniziale non nulla, percorrendo una traiettoria parabolica.

Il codice seguente illustra il linguaggio risultato dell'idea di componente: per prima cosa viene creato un oggetto **Rettangolo** chiamato `r`, dopodiché viene installato in un nuovo oggetto **Assembly** attraverso il metodo `install()`.

Il dispiegamento multiplo dell'unico oggetto contenuto nel gruppo è effettuato all'interno di un ciclo iterativo in cui sono calcolate le coordinate dell'oggetto attratto dalla gravità in funzione del tempo t a partire dall'istante 0, chiamando il metodo `deploy()`. Per stampare il disegno del gruppo basterà chiamare il metodo `draw()` dell'oggetto **Assembly**:

```
-- creazione rettangolo
r = Rectangle:new{x=1.2,y=0.8}

-- creazione componente
a = Assembly:new()

-- installazione del rettangolo
a:install(r)

-- parametri cinematici
local g = 9.80665 -- m/s^2
local vx, vy = 8, 0 -- m/s

for t=0, 2.5, 0.10 do
    -- calcolo coordinate
    local x = t*vx
    local y = -0.5*g*t^2+t*vy
    -- dispiegamento oggetto
    a:deploy(r, {x=x,y=y})
end

-- disegno del componente
a:draw()
```

L'implementazione della classe `Assembly` è piuttosto semplice grazie alle caratteristiche della tabella di Lua: possiamo infatti utilizzare riferimenti di tabella come chiave di campi valore. Si prevede di utilizzare una tabella chiamata `content` interna all'oggetto le cui chiavi sono i riferimenti agli oggetti grafici installabili nel componente. A queste chiavi riferimento si assocerà una tabella che a sua volta contiene la lista di tabelle ciascuna relativa alle informazioni di dispiegamento (per brevità, nel codice sono stati omessi i controlli di validità):

```
-- oggetto Assembly
Assembly = {} -- init
Assembly.classname = 'Assembly'
Assembly.__index = Assembly -- metametodo
Assembly.version = '0.01' -- version number
Assembly.date = '20120915' -- library date

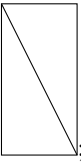
-- costruttore
function Assembly:new()
    local a = {}
    -- contenitore per oggetti
    -- e loro disposizione
    a.content = {}

    setmetatable(a, self)
    return a
end

-- caricamento oggetto
function Assembly:install(o)
    if not self.content[o] then
        self.content[o] = {}
    end
end

-- informazione di posizione
-- o : a graphic object
-- tdeploy : tables list of {x, y, angle}
function Assembly:deploy(o, tdeploy)
    if self.content[o] then
        local tpos = self.content[o]
        tpos[#tpos+1] = tdeploy
    end
end
```

In questo momento il rettangolo appare così: 

Ma può diventare anche più grande: 

...e di nuovo piccolo: 

FIGURA 5: Dimostrazione che gli oggetti grafici, una volta creati come riferimento globale, possono essere riutilizzati più volte ed ovunque nel documento, chiamandone i metodi desiderati in un ambiente `gol`.

```
-- metodo di disegno
function Assembly:draw()
    for o, tdeploy in pairs(self.content) do
        for _, pos in pairs(tdeploy) do
            local x, y = pos.x, pos.y
            local a = pos.angle
            o:draw{dx = x, dy = y, angle = a}
        end
    end
end
```

4.7 Riutilizzo delle figure

Gli oggetti creati come variabili globali, senza cioè che sia premessa nell'assegnazione la parola chiave `local`, rimangono nella memoria e quindi possono essere riutilizzati in altre posizioni del sorgente, così anche per gli oggetti `GOL` creati nel documento ed è quindi possibile chiamarne i metodi.

Nel successivo esempio un rettangolo e una delle sue diagonali vengono create all'inizio del documento per poi essere disegnate più volte modificandone le dimensioni, come dimostra la figura 5.

Nel codice dell'esempio, riportato di seguito, si può notare come la modifica delle dimensioni del rettangolo `x` ed `y` avviene in modo diretto sui campi dell'oggetto in forma di *dot notation* e, da quanto detto precedentemente sui principi della programmazione ad oggetti, ciò non risulta corretto. Infatti, nello sviluppo della libreria potrebbero cambiare i nomi delle variabili in cui sono memorizzate le dimensioni del rettangolo, o potrebbe cambiare il tipo di dato che le rappresenta.

Lo sviluppatore è tenuto a mantenere immutata solo la definizione dell'interfaccia. Nel caso descritto, il metodo `setxy()` continuerebbe comunque a funzionare per la modifica delle dimensioni del rettangolo, ma non è detto che valga la stessa cosa per l'assegnazione diretta di larghezza e altezza.

In sintesi, non accedere direttamente ai campi interni dell'oggetto è una buona regola che può essere violata in Lua, ma il codice mostra come sarebbe concettualmente più semplice e diretto agire su di essi, tanto che questi campi potrebbero rientrare nell'interfaccia dell'oggetto anziché rimanere pseudo-privati. Una scelta importante questa perché coinvolge i concetti base del linguaggio.

```
\begin{gol}
r = Rectangle:new{x=0.25,y=0.5}
a = r:getPoint{vertex='nw'}
b = r:getPoint{vertex='se'}
```

```
l = Line:new{a,b}
\end{gol}
```

In questo momento il rettangolo appare così:

```
\begin{gol}
r:draw()
l:draw()
\end{gol}
```

Ma può diventare anche più grande:

```
\begin{gol}
r.x, r.y = 1, 2
```

```
r:draw()
l:draw()
\end{gol}
;
```

\dots e di nuovo piccolo:

```
\begin{gol}
r.x, r.y = 1, 0.5
```

```
r:draw()
l:draw()
\end{gol}.
```

4.8 Il codice nascosto

Il codice TikZ che viene *iniettato* in T_EX dai metodi `draw()` degli oggetti grafici creati dall'utente, è completamente inaccessibile, ma non è difficile modificare questo comportamento memorizzando in una tabella il codice nascosto prodotto man mano per poi salvarlo in un file esterno.

Questo aspetto mette in luce come il progetto della libreria sia poco modulare. Occorrerebbe scindere il codice nascosto dai metodi degli oggetti perché questo renderebbe più semplice lo sviluppo ed allo stesso tempo consentirebbe di aggiungere ulteriori motori grafici intercambiabili. Potremmo così ottenere il codice nascosto anche in formato METAPOST per esempio, od anche in formato vettoriale `dxg` (AUTODESK, 2011), formato usato nei disegni CAD.

4.9 Nuovi pacchetti

Nella sezione precedente si è accennato ad una struttura modulare più razionale per la libreria GOL per separare il codice degli oggetti geometrici da quello effettivamente necessario per disegnarli secondo un formato grafico sottostante, facendo così intravedere uno dei punti più importanti dell'introduzione di LuaT_EX: la possibilità di sviluppare progetti dalle dimensioni molto superiori a quelli limitati all'uso del solo motore tipografico T_EX.

Così, la libreria GOL potrebbe favorire la scrittura di librerie specifiche da parte di contributori esterni allo sviluppo di base, proprio come avviene per il pacchetto PGF, fornendo agli oggetti metodi

per caricare in essi nuove funzioni e l'infrastruttura per costruire disegni complessi pronti per essere utilizzati e ricchi di proprietà grafiche modificabili dall'utente.

5 Ereditarietà

Il meccanismo dell'ereditarietà, caratteristica essenziale della programmazione ad oggetti, consiste nel creare una nuova classe a partire da una già esistente per estenderne e modificarne le funzionalità. Con la classe padre si rappresenta un concetto più generale, che è reso più specifico e dettagliato dalla classe figlia.

Ogni processo di ereditarietà crea una nuova relazione tra concetti per costruire l'intera struttura che rappresenta un particolare problema reale. Nel nostro caso l'ereditarietà può essere impiegata in tre contesti diversi: quello dello sviluppo della libreria in cui opera il programmatore, quello dello sviluppo di pacchetti d'estensione, e quello del singolo documento T_EX in cui opera l'utente finale.

La libreria GOL potrebbe implementare una struttura di relazioni tra oggetti grafici a partire da una classe base ottenendo maggiore chiarezza del codice e migliore possibilità di sviluppo.

5.1 Livelli d'astrazione

Tutti gli oggetti grafici fanno uso di un sistema di riferimento locale, in particolare delle coordinate dell'origine chiamate *punto di ancoraggio*, e di alcune proprietà di disegno come il colore del tratto e del riempimento, il tipo di linea e il suo spessore. Così è naturale pensare di creare una classe base da cui ereditare per costruire tutti gli oggetti grafici reali come cerchi e linee.

Questa classe base, detta **Element**, disporrà di un metodo per la modifica delle coordinate del punto di ancoraggio e di alcuni metodi d'interfaccia privi d'implementazione che obbligatoriamente le classi derivate devono definire, come per esempio il metodo `draw()`. Lua non dispone del supporto per la creazione di questi *metodi virtuali*. Possiamo solamente inserire nei metodi dell'interfaccia un'istruzione di errore che interrompe l'esecuzione e la compilazione del documento avvertendo che il metodo reale non è stato correttamente definito nella classe derivata.

Quanto al costruttore, la classe base non ne può disporre poiché non si tratta di un oggetto reale. Valgono per esso le stesse considerazioni fatte precedentemente per qualsiasi altro metodo virtuale della classe.

Lua non dispone nemmeno di un supporto esplicito all'ereditarietà che può quindi essere implementata in diversi modi. Quello più semplice consiste nell'indurre il compilatore a percorrere all'indietro la gerarchia delle classi nel tentativo di ricerca-

re il componente chiamato attraverso la funzione `__index()` e le metatabelle.

Per esemplificare il comportamento del compilatore si consideri la classe `Rectangle` che questa volta estende la classe base `Element` dotata di un metodo `setFillColor()` per impostare il colore di riempimento degli oggetti grafici:

```
r = Rectangle:new{}
r:setFillColor('orange')
```

Poiché il metodo `setFillColor()` non è definito in `Rectangle`, Lua cerca il metodo `__index()` nella metatabella associata ad `r` — il riferimento a `Rectangle` stesso. Il metodo non è presente pertanto si ricade nella stessa situazione: Lua cerca la metatabella associata a `Rectangle` che questa volta è un riferimento ad `Element`, così in conclusione viene eseguito il metodo `setFillColor()` della superclasse con un primo parametro `self` che fa riferimento ad `r` ed un secondo argomento contenente il valore del colore.

L'effetto di questa ricerca a cascata è quello per cui la classe derivata *eredita* tutti i campi ed i metodi della classe base. In definitiva, ciò che accade nel codice precedente è la creazione — od eventualmente la modifica — del campo interno ad `r` che memorizza il colore di riempimento attraverso il metodo implementato nella classe base. Per convincerci di questo è sufficiente leggere il codice del metodo `setFillColor()` ricordando gli argomenti che esso riceve — `self` ovvero `r` ed `'orange'` — al momento dell'esecuzione:

```
function Element:setFillColor(col)
    if col then self.fillcol = col end
end
```

In generale, quando un metodo di un oggetto richiede la lettura di un campo valore o di un campo funzione della superclasse, la struttura della tabella che lo implementa non viene modificata mentre se l'oggetto richiede la scrittura degli stessi campi, la prima volta che questo accade sarà creata una copia locale che da quel momento diviene il nuovo campo interno all'oggetto in lettura/scrittura.

Dal punto di vista dell'utente della libreria ad oggetti, l'effetto della scrittura di un campo valore viene percepito semplicemente come la modifica di un elemento che concettualmente fa già parte dello stato dell'oggetto. Invece, la scrittura di un campo metodo è intesa come applicazione del *polimorfismo*, modifica il comportamento dell'oggetto con la sostituzione del metodo ereditato dalla superclasse.

In altre parole, in relazione alla modifica di campi e di metodi di una classe derivata, lo sviluppatore Lua osserva una stessa identica procedura seguita dal compilatore, mentre lo sviluppatore che adotta il paradigma ad oggetti sperimenta per essi concetti profondamente diversi.

Nella figura 6 è riportato un disegno di verifica della correttezza della nuova implementazione che fa derivare la classe `Rectangle` dalla classe

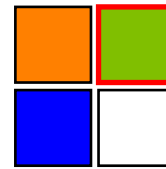


FIGURA 6: Un primo esempio per la verifica del corretto funzionamento dell'oggetto `Rectangle`, ottenuto come estensione di una classe base con il meccanismo dell'ereditarietà. Tutti gli oggetti derivati da essa condividono un insieme di funzionalità comuni senza bisogno di riscrivere codice.

`Element`. Nel relativo codice riportato di seguito vengono istanziati due oggetti `Rectangle` modificandone le proprietà di colore. La verifica ha esito positivo poiché ciascun oggetto mantiene correttamente i propri parametri perfettamente separati da quelli degli altri.

```
\begin{goll}
r = Rectangle:new{}

r:setFillColor('blue')
r:draw()

q = Rectangle:new{}
q:draw{dx=1.1}
q:setFillColor('orange')
q:draw{dy=1.1}

q:setLinewidth(2)
q:setFillColor('orange!50!green')
q:setLineColor('red')
q:draw{dx=1.1, dy=1.1}
\end{goll}
```

Gli elementi essenziali dell'implementazione sono riportati nei seguenti frammenti di codice. La classe `Element` non ha un costruttore ed è priva anche di un metodo `draw()` — che dovrebbe essere inteso come un metodo virtuale — quando invece crea una serie di campi che saranno comuni a tutti gli oggetti delle classi derivate da essa. Si noti come una delle prime assegnazioni riguardi il metodo `__index()` valorizzato con il riferimento alla stessa tabella `Element`, istruzione questa indispensabile per il corretto funzionamento della ricerca dei metodi da ereditare:

```
-- oggetto base 'Element'
Element = {} -- init
Element.__index = Element -- metametodo

Element.classname = 'Element' -- class id
Element.version = '0.01' -- version number
Element.date = '20120917' -- library date

-- campi che saranno comuni a tutte le classi derivate
-- :coordinante del punto di ancoraggio
Element.x = 0
Element.y = 0

-- :proprietà grafiche
Element.lw = 1 -- pt
Element.linecol = nil -- means 'black'
Element.fillcol = nil -- means 'trasparent'
```

```
-- metodo d'impostazione dell' 'origine locale' oggetto
-- :x,y -> absolute coordinates
-- :dx, dy -> relative coordinates
function Element:setAnchor(t)
  if t then
    if t.x then
      if t.dx then error("Mix mode not allowed.")
      end
      self.x = t.x
    end
    if t.y then
      if t.dy then error("Mix mode not allowed.")
      end
      self.y = t.y
    end
    if t.dx then self.x = self.x + t.dx end
    if t.dy then self.y = self.y + t.dy end
  end
end

-- set property functions
function Element:setLineWidth(lw)
  if lw then self.lw = lw end
end
function Element:setLineColor(col)
  if col then self.linecol = col end
end
function Element:setFillColor(col)
  if col then self.fillcol = col end
end
```

Le righe di codice indispensabili per comprendere il meccanismo dovuto alle metatabelle sono le seguenti, riguardanti l'implementazione della classe derivata `Rectangle`. Le istruzioni fondamentali che vi compaiono sono l'assegnazione di `Element` come metatabella di `Rectangle` e la valorizzazione del metodo `__index()` con il riferimento alla tabella stessa. Nel costruttore è necessario impostare anche la metatabella degli oggetti istanza per completare le informazioni necessarie al compilatore Lua ed implementare così la derivazione:

```
-- oggetto derivato 'Rectangle'
Rectangle = {} -- init

-- metatabella classe figlia = superclasse
setmetatable(Rectangle, Element)
Rectangle.__index = Rectangle
Rectangle.classname = 'Rectangle'
Rectangle.version = '0.04' -- version number
Rectangle.date = '20120917' -- version date

-- costruttore
function Rectangle:new(t)
  local o = {}
  if t then
    o.xdim = t.x or 1
    o.ydim = t.y or 1
  else
    o.xdim, o.ydim = 1, 1
  end
  o.refpoints = {}

  setmetatable(o, self)
  return o
end

...
<campi e metodi della classe derivata>
```

5.2 Creare nuovi oggetti

Nulla vieta che anche l'utente possa creare nuovi oggetti ereditando da classi esistenti. Dovrà tuttavia rispettare l'obbligo di implementare alcuni metodi fondamentali, caratteristica propria dell'oggetto grafico che egli intende costruire.

Per facilitare la specializzazione degli oggetti già disponibili, la libreria `GOL` dovrebbe mettere a disposizione un metodo che si occupi dei dettagli come l'assegnazione delle metatabelle e dei metametodi.

6 Disegno parametrico

Il *disegno parametrico* è una funzionalità potente, ma complessa da utilizzare, che fa dipendere la geometria e l'aspetto di un disegno da valori indipendenti e modificabili. Questi parametri possono essere grandezze relative alla geometria di oggetti già esistenti come la lunghezza di un lato o quella del diametro di un cerchio.

Avvalersi del disegno parametrico significa poter *vincolare* le caratteristiche di un'entità a quelle di altre, per rappresentare relazioni geometriche e mettere in pratica i concetti sottostanti alla costruzione grafica. Risulterà molto efficiente, una volta stabilito il vincolo, modificare il disegno solamente nelle proprietà indipendenti.

Per esempio, è possibile vincolare il lato di un rettangolo ad essere uguale all'altro, creando un quadrato, oppure è possibile far dipendere la misura di un diametro da un parametro numerico, od ancora regolare un angolo ad assumere sempre un determinato valore.

Essendo scritta in Lua, la libreria `GOL` può effettivamente tentare di offrire funzionalità di disegno parametrico attraverso una nuova classe denominata `Param` che memorizza il valore del parametro e lo rende vincolo per gli oggetti grafici.

Dove per un argomento ci si aspetta una misura si può invece fornire un oggetto `Param`. Nel codice si potrebbe aggiungere un controllo sul tipo oppure trasformare ogni campo in una funzione che restituisce il valore oppure la corrispondente funzione di interrogazione del parametro.

7 Organizzare il disegno

Allo stesso modo dei disegni CAD ed anche di TikZ, il potente pacchetto per la grafica vettoriale in `TeX`, è possibile organizzare gli elementi grafici attribuendoli ad un dato insieme chiamato *layer*. L'idea assomiglia al modo in cui il disegnatore sovrappone tanti fogli trasparenti ciascuno con diverse porzioni grafiche, per comporre il disegno complessivo.

Un layer può essere *spento* così che i suoi oggetti non compariranno più nel disegno — per esempio per ottenere una stampa ad una scala più piccola eliminando i dettagli — ed essere caratterizzato

da uno *stile* ovvero le proprietà di colore, spessore e tipo di linea, che gli oggetti del layer possono condividere.

Cambiando le proprietà del layer cambiano di conseguenza le corrispondenti proprietà di tutti gli oggetti in esso contenuti, salvo che questi ne abbiano di proprie. Per esempio, il colore di tracciamento di un oggetto potrebbe assumere il valore di `DaLayer` assumendo quindi il parametro stabilito per il layer oppure, pur appartenendo al layer, avere una proprietà di colore personalizzata.

Fino ad ora l'elemento `Layer` ha due principali caratteristiche: memorizza *proprietà* grafiche per gli oggetti ad esso assegnati e risulta che un oggetto può appartenere ad un unico layer. Ciò significa che i layer adottano una struttura ad un unico livello.

Conseguenza della prima caratteristica è che l'elemento `Layer` non è un oggetto e pertanto apparirà all'utente come una classe non istanziabile dotata unicamente di *metodi statici*. Nella sintassi del linguaggio Lua significa che i metodi verranno chiamati con la notazione punto, e non compariranno in istruzioni di assegnazione ma in espressioni semplici, come nel seguente listato frammento di un ambiente `gol`:

```
-- creazione di un nuovo layer
-- con un metodo statico
Layer.add{name      = '20',
           linecolor = 'blue!50',
           fillcolor = 'yellow',
           linewidth = 0.2,
           linetype  = 'dashed',
}

-- spegnimento di un layer
Layer.off{'30'}
```

```
-- cancellazione di un layer
Layer.delete{'details'}
```

La seconda caratteristica porta invece ad estendere il concetto base di *layering* dei programmi CAD, in una struttura a più livelli strutturata ad albero: un layer può contenere altri sotto-layer. Un oggetto raggruppato in un layer può essere influenzato dalle proprietà associate ad un layer di livello superiore. Il livello di complessità organizzativa del disegno in layer a più livelli rimane ad esclusiva scelta dell'utente che può ignorare completamente a quale layer appartengono gli oggetti, o può invece creare un unico livello, oppure ancora può avvalersi di livelli multipli di organizzazione.

8 Conclusioni

La libreria grafica `GOL` descritta nell'articolo e sviluppata per sperimentare la programmazione ad oggetti da parte degli utenti finali del sistema `TEX` dimostra che è possibile progettare linguaggi con elevata intuitività grazie alla rappresentazione diretta di concetti.

Il linguaggio separa la definizione delle entità geometriche dal loro effettivo disegno consentendo all'utente di modificarne le proprietà a piacimento, di avvalersi di sofisticati metodi di costruzione e di riutilizzare i disegni nel documento.

Naturalmente è possibile applicare la stessa idea in molti altri settori della tipografia asincrona con `TEX`. Si pensi alla generazione di report in forma di tabelle compresi quelli dalla struttura molto complessa e con informazioni residenti in database esterni o memorizzati in file nei formati più vari, oppure ad un sistema di controllo dei parametri tipografici del documento.

L'evoluzione del sistema `TEX` continua in modo inaspettato ed originale per consentire di raggiungere un più elevato numero di utenti e migliori risultati tipografici. `LATEX3`, `ConTEXt-mkiv`, `LuaLATEX` e nuovi pacchetti basati su Lua prospettano un panorama applicativo quanto mai vasto ed in frenetico sviluppo.

Tuttavia non è mai facile concretizzare le idee e le potenzialità perché il linguaggio dovrà apparire all'utente il più possibile coerente ed equilibrato e garantire nel tempo il rapido e sicuro apprendimento da parte degli utenti ed il non meno importante fattore evolutivo delle applicazioni.

L'idea che l'utente scriva codice ad oggetti in un sorgente `TEX` è una recente possibilità dovuta al nuovo motore `LuaTEX`. Altrettanti recenti sviluppi dei linguaggi di programmazione orientati per un mondo *completamente* basato sulla rete internet — si pensi ad esempio ai sistemi operativi basati su tecnologie *cloud* ed al linguaggio `Go` di Google (GRIESEMER *et al.*, 2012) — smorzano un po' l'entusiasmo per la programmazione ad oggetti. La complessità di progettazione della OOP e la riduzione dell'efficienza in fase di esecuzione del codice sono forse i motivi principali che hanno fatto sì che questo paradigma non abbia raggiunto pienamente tutti gli obiettivi iniziali.

Si tratta del fondamentale processo di evoluzione indotto da nuove esigenze derivate a sua volta dallo spostamento delle attività di elaborazione dati dalla tradizionale *scrivania* ad un *luogo non fisico*, che nello sviluppo tende a semplificare le gerarchie nelle strutture dei linguaggi ad oggetti ed a favorire il raggiungimento delle massime prestazioni sfruttando processori a più core con l'esecuzione parallela e contemporaneamente a favorire l'aumento della produttività degli sviluppatori.

Nel mondo `TEX`, dove il linguaggio è lo strumento essenziale degli utenti, è sempre più importante affiancare ad una elevata qualità tipografica un'elevata produttività degli utilizzatori. La programmazione ad oggetti è un sicuro tema da esplorare per raggiungerla.

9 Ringraziamenti

Ringrazio il revisore anonimo e Gianluca Pignalberi direttore di *Ar_sTEXnica* che hanno fatto pienamente parte della *squadra* che ha portato questo articolo fino alla conclusione, squadra di cui fa certamente parte anche Agostino De Marco che ha più volte contribuito al testo con suggerimenti e correzioni e che è sicuramente pronto per il nostro prossimo lavoro. Grazie a tutti.

Ringrazio in modo particolare anche la mia famiglia (lo devo fare... altrimenti sono guai) perché mi sostiene nella scoperta di nuovi paesaggi del mondo _{TEX}.

Riferimenti bibliografici

AUTODESK (2011). *DXF Reference*. Autodesk Inc.

CORNELL, G. e HORSTMANN, C. S. (1997). *Core Java 2 Fondamenti*. Professionale. Pearson Education Italia, 7^a edizione.

GRIESEMER, R., PIKE, R. e THOMPSON, K. (2012). «Go programming language». Sito web ufficiale <http://golang.org/>.

IERUSALIMSKY, R. (2006). *Programming in Lua*. Lua.org, 2^a edizione.

LOWAGIE, B. e ALTRI (2012). «Libreria PDF in Java o C#». Sito web <http://itextpdf.com/>.

PÈGOURIÈ-GONNARD, M. (2012). «The luacode package version 1.2a». Disponibile sul sito web www.ctan.org.

TANTAU, T. (2010). «PGF manual version 2.10». Disponibile su www.ctan.org.

▷ Roberto Giacomelli
Carrara (MS)
giaconet dot mailbox at gmail
dot com

MFLua: Instrumentation of METAFONT with Lua

Luigi Scarso

Abstract

We present MFLua, a METAFONT version which is capable of code instrumentation and has an embedded Lua interpreter that allows glyphs curves extraction and post-processing. We also show and discuss an example of a METAFONT source processed by MFLua to output an OpenType font.

Sommario

Presentiamo MFLua, una versione di METAFONT che permette la strumentazione del codice e dotata di un interprete Lua per consentire l'estrazione ed il post-processing delle curve di un glifo. Mostriamo e discutiamo anche un esempio di font OpenType prodotto da un sorgente METAFONT.

1 Introduction

MFLua is a version of METAFONT, Knuth's program (KNUTH, 1986b) designed to draw fonts. MFLua has an embedded Lua interpreter, as well as the capability of the Pascal-WEB code instrumentation to output information about bitmaps and curves used in glyphs drawing. The latest capability is known as *code tracing*. MFLua's main goal is to ease the production of vector fonts which source code is a METAFONT code. MFLua doesn't extend the METAFONT language in any way (i.e., it's not possible to embed Lua code in a METAFONT source file), so that a METAFONT source file is fully compatible with MFLua and vice versa. MFLua won't be extended like LuaTeX extends pdfL^AT_EX. The code instrumentation is a facility to gather and manage information collected in the log file when METAFONT *tracing* instructions are enabled. MFLua automatically saves data into Lua tables using external Lua scripts. Therefore a programmer can manage these tables according to his needs, i.e. extracting a glyph vector outline(s). Rephrasing the previous statements, MFLua is a (bitmap) *tracing* program that knows curves in advance instead of determining them from the bitmap. Please notice that this is only possible when the data have been gathered.

The paper has the following structure: after explaining what *code instrumentation* is (section 2), it shows the components used to trace a glyph (section 3) and finally two different approaches to manage curves (section 4).

As a final remark, we consider MFLua as being in a state between a proof of concept and alpha and

it's not (yet) too user-friendly. Its code is hosted at <https://github.com/luigiScarso/mflua>.

2 Code instrumentation

METAFONT is written in Pascal-WEB (a programming language by Donald Knuth to have a real literate programming tool. As the name suggests, it's a subset of Pascal) and is automatically translated into C by *tangle* and *web2c*. Instrumentation is the capability to add *trace statements* (a.k.a. *sensors*) in strategic points of the code to register current state information and pass it to the Lua interpreter. A typical sensor has the *mflua* prefix. We can see some sensors in the following chunk of code, the main body of METAFONT (slightly refolded to fit the printing area).

```
@p begin @!{|start_here|}
mflua_begin_program;
{in case we quit during initialization}
history:=fatal_error_stop;
t_open_out; {open the terminal for output}
if ready_already=314159 then goto start_of_MF;
@<Check the 'constant' values...@>;
if bad>0 then
  begin wterm_ln('Ouch---my internal constants
    have been clobbered!',
    '---case ',bad:1);
@.Ouch...clobbered@>
  goto final_end;
end;
{set global variables to their starting values}
initialize;
@!init if not get_strings_started then
  goto final_end;
init_tab; {initialize the tables}
init_prim; {call |primitive| for each primitive}
init_str_ptr:=str_ptr; init_pool_ptr:=pool_ptr;@/
max_str_ptr:=str_ptr; max_pool_ptr:=pool_ptr;
fix_date_and_time;
tini@/
ready_already:=314159;
mfluaPRE_start_of_MF;
start_of_MF: @<Initialize the output routines@>;
@<Get the first line of input and prepare
  to start@>;
history:=spotless; {ready to go!}
mflua_initialize;
if start_sym>0 then
  {insert the '\&{everyjob}' symbol}
  begin cur_sym:=start_sym; back_input;
  end;
mfluaPRE_main_control;
main_control; {come to life}
mfluaPOST_main_control;
final_cleanup; {prepare for death}
mfluaPOST_final_cleanup;
end_of_MF: close_files_and_terminate;
final_end: ready_already:=0;
end.
```

We're going to examine the role of the `mflua_begin_program` sensor. The Pascal-into-C translator, `web2c`, is smart enough to distinguish a symbol already present in the Pascal source from an external symbol (i.e., a symbol defined in another file). In the latter case, the programmer has to register that symbol into the file `texmf.defines` if the symbol is related to an argumented procedure: the translator will manage properly the arguments translation. The translated code will contain the C form of the sensor symbol, which will be resolved at compile-time — i.e., we need an object file that contains that symbol. Each sensor is stored into `mflua.h` and `mflua.c`. The first one lists the symbol:

```
#include "lua51/lua.h"
#include "lua51/lualib.h"
#include "lua51/lauxlib.h"
#include <kpathsea/c-proto.h>
#include <web2c/config.h>

extern lua_State* Luas[];
extern int mfluabeginprogram();
```

while the second one contains the corresponding function source code:

```
int mfluabeginprogram()
{
    lua_State *L = luaL_newstate();
    luaL_openlibs(L);
    Luas[0] = L;
    /* execute Lua external "begin_program.lua" */
    const char* file = "begin_program.lua";
    int res = luaL_loadfile(L, file);
    if ( res==0 ) {
        res = lua_pcall(L, 0, 0, 0);
    }
    priv_lua_reporterrors(L, res);
    return 0;
}
```

The above function initializes the Lua interpreter, stores its state in the array `Luas[]` (it would be possible to have more than one interpreter but this feature is currently neglected) and then executes the external script `begin_program.lua` calling `lua_pcall(L, 0, 0, 0)`. This call protects the interpreter from errors. Every time we run `mf` (the METAFONT program), it loads and executes the Lua script `begin_program.lua`, customizable by programmers.

We surely need to pay attention to some issues. The first one is that literate programming style allows to collect every changes we make in a source file into a *change* file (`mf.ch` in our case), which is then merged into a Pascal program by `tangle`. This means that inserting a sensor can interfere with the change file. In this case we also have to insert the sensor into the change file as we do, for instance, with `mfluaPRE_make_ellipse(major_axis, minor_axis, theta, tx, ty, 0)`. Of course the right solution is directly inserting the sensors in

the change file. Unfortunately it's usually faster discovering where to insert a sensor in the source file and then managing conflicts in the change file: source files have a context — the source itself — that change file don't. The second issue is the need to export some METAFONT variables and constants to the Lua interpreter. An easy way to accomplish this task is inspecting the translated C code to realize how those variables and constants are managed (e.g., to make Lua read the `charcode` variable, which contains the index of the current glyph, we need to know that it's stored into the `internal` array at index 18 (the index is from the METAFONT WEB source) so that we can write a wrapper function like the following one:

```
#define roundunscaled(i) (((i>>15)+1)>>1)
EXTERN scaled internal[maxinternal + 1] ;
static int
priv_mfweb_LUAGLOBALGET_char_code(lua_State *L)
{
    integer char_code=18;
    integer p=roundunscaled(internal[char_code])%256;
    lua_pushnumber(L,p);
    return 1;
}
```

and then make it available to the Lua interpreter as the `LUAGLOBALGET_char_code` variable:

```
int mfluainitialize()
{
    /* execute Lua external "mfluaini.lua" */
    lua_State *L = Luas[0];
    /* register lua functions */
    :
    lua_pushcfunction(L,
        priv_mfweb_LUAGLOBALGET_char_code);
    lua_setglobal(L, "LUAGLOBALGET_char_code");
    :

    /* execute Lua external "mfluaini.lua" */
    const char* file = "mfluaini.lua";
    int res = luaL_loadfile(L, file);
    if ( res==0 ) {
        res = lua_pcall(L, 0, 0, 0);
    }
    priv_lua_reporterrors(L, res);
    return 0;
}
```

Users can read and set this variable though the set value won't be passed to METAFONT in order to interfere as little as possible with its state. That's why we prefer to inspect the translated C code, which quality depends on the translation performed at compile-time. A clean solution should only depend on the WEB source. For historical reasons, translating code from Pascal into C outputs two files (`mf0.c` and `mf1.c`). Finding a symbol implies searching in two files, which hardens the process.

There are currently 24 sensors, 33 global variables and 15 scripts, though it's possible to increase these quantities if we discover that tracing a specific function inside METAFONT is better than

reimplementing it in Lua. While it's easy to implement the algorithm to draw a curve in Lua, it's slightly harder to implement the algorithm to fill a region. Whatever, the main goal is to keep the number of sensors as low as possible. Notice that MFLua currently reads scripts from the current folder and doesn't use the standard T_EX folders.

The counterpart of `mflua_begin_program` is `mflua_end_program`, which calls the `end_program.lua` script. It contains all the functions used to transform the components of a glyph, the subject of the next section.

3 The components of a glyph

METAFONT mainly manages Bézier cubic curves (see fig. 1).¹ This curve is completely described by four *controls points*: **p** (called the *starting point*), **c**₁, **c**₂ and **q** (known as the *ending point*). The METAFONT command to draw such a curve is

```
draw p .. controls c1 and c2 .. q;
```

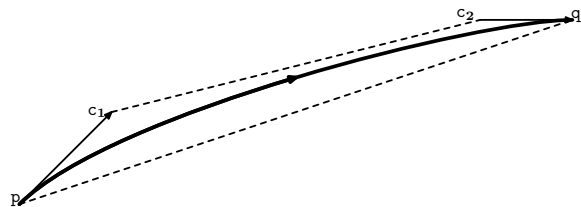


FIGURE 1: A cubic Bézier curve and its convex hull.

This curve lies on the *XY* plane and its *parametric form* is quite simple:

$$\mathbf{B}(t) = (1-t)^3\mathbf{p} + 3(1-t)^2t\mathbf{c}_1 + 3(1-t)t^2\mathbf{c}_2 + t^3\mathbf{q} \quad \forall t \in [0, 1] \quad (1)$$

The corresponding algebraic expression, the *closed form*, is more complex but it's useful to quickly test whether a point belongs to the curve or not.

Equation (1) has first derivatives $\mathbf{B}'(0) = 3(\mathbf{c}_1 - \mathbf{p})$ and $\mathbf{B}'(1) = 3(\mathbf{q} - \mathbf{c}_2)$ respectively when $t = 0$ and $t = 1$. We can easily calculate them when we know **p**, **c**₁, **c**₂ and **q**. An important property is that a Bézier cubic curve is completely contained in the polygon **p c**₁ **c**₂ **q p** (the convex hull) and this immediately leads to the conclusion that the intersection of two curves is empty if and only if the intersection of their convex hulls is empty. Another important property is the existence of the *De Casteljau's algorithm*, very easy to implement: given the four control points of a curve and a time t_1 , it returns the point $(x_1, y_1) = \mathbf{B}(t_1)$ on the curve and the two series of control points, one for the curve $\mathbf{B}_l(t) = \mathbf{B}(t), t \in [0, t_1]$ (the *left*

side) and one for the curve $\mathbf{B}_r(t) = \mathbf{B}(t), t \in [t_1, 1]$ (the *right side*). It recursively reduces the curve $\mathbf{B}(t), t \in [0, 1]$ tracing to the tracing of the left side $\mathbf{B}_l(t) = \mathbf{B}(t), t \in [0, 1/2]$ and the right side $\mathbf{B}_r(t) = \mathbf{B}(t), t \in [1/2, 1]$ (the recursion ends when the distance between two points (x_j, y_j) and (x_{j+1}, y_{j+1}) is less than a pixel).

The De Casteljau's algorithm is useful because it estimates how long a curve is counting the number of pixels covered by the curve. It also finds intersections of two curves $\mathbf{B}(t)$ and $\mathbf{C}(t)$ reducing this problem to the problem of calculating the intersection of four curves: left and right side of $\mathbf{B}(t)$ and left and right side of $\mathbf{C}(t)$ for $t = 1/2$. The algorithm keeps working when one curve degenerates into a segment (i.e., if $\mathbf{p} = \mathbf{c}_1$ and $\mathbf{c}_2 = \mathbf{q}$) or when it degenerates into a point ($\mathbf{p} = \mathbf{c}_1 = \mathbf{c}_2 = \mathbf{q}$). Therefore it can be used to find an intersection between a line and a curve and to test if a point belongs to a curve. A set of curves $\{\mathbf{B}_1, \mathbf{B}_2 \dots \mathbf{B}_n\}$ where $\mathbf{q}_{j-1} = \mathbf{p}_j$ and $\mathbf{q}_n = \mathbf{p}_0$ is a simple cycle if the only intersection is $(x, y) = \mathbf{q}_n = \mathbf{p}_0$. Simple cycles are the building blocks of a glyph: a simple cycle can be filled or unfilled and, according to METAFONT's point of view, a glyph is a set of cycles filled and/or unfilled at the right moment.

A normal METAFONT designer doesn't care about these details because METAFONT has a high level language to describe curves, points, lines, intersections, filled and unfilled cycles and, most important, pens. The listed entities produce a combination of two different basic draws: regions (un)filled by a *contour* and regions (un)filled by the stroke of a pen, i.e., the *envelope of a pen*. Both are simple cycles, but their origin is very different.

Let's consider the code of the glyph 0 from the file `xmssdc10.mf`:

```
cmchar "The numeral 0";
beginchar("0",9u#,fig_height#,0);
italcorr fig_height#*slant-.5u#;
adjust_fit(0,0);
penpos1(vair,90);
penpos3(vair,-90);
penpos2(curve,180);
penpos4(curve,0);
if not monospace:
  interim superness:=sqrt(more_super*hein_super);
fi
x2r=hround max(.7u,1.45u-.5curve);
x4r=w-x2r; x1=x3=.5w;
y1r=h+o; y3r=-o;
y2=y4=.5h-vair_corr;
y2l:=y4l:=.52h;
penstroke pulled_arc.e(1,2) & pulled_arc.e(2,3)
& pulled_arc.e(3,4)
& pulled_arc.e(4,1) & cycle; % bowl
penlabels(1,2,3,4);
endchar;
```

Fig. 2 shows a glyph only made by two contours which are the result of `penpos` and `penstroke` macros. Of course we could obtain the same result drawing 24 curve sections (12 for the outer contour,

1. I borrow notation from MARSH (2005), where points and functions in the Bézier curves section are represented by bold, upright letters.

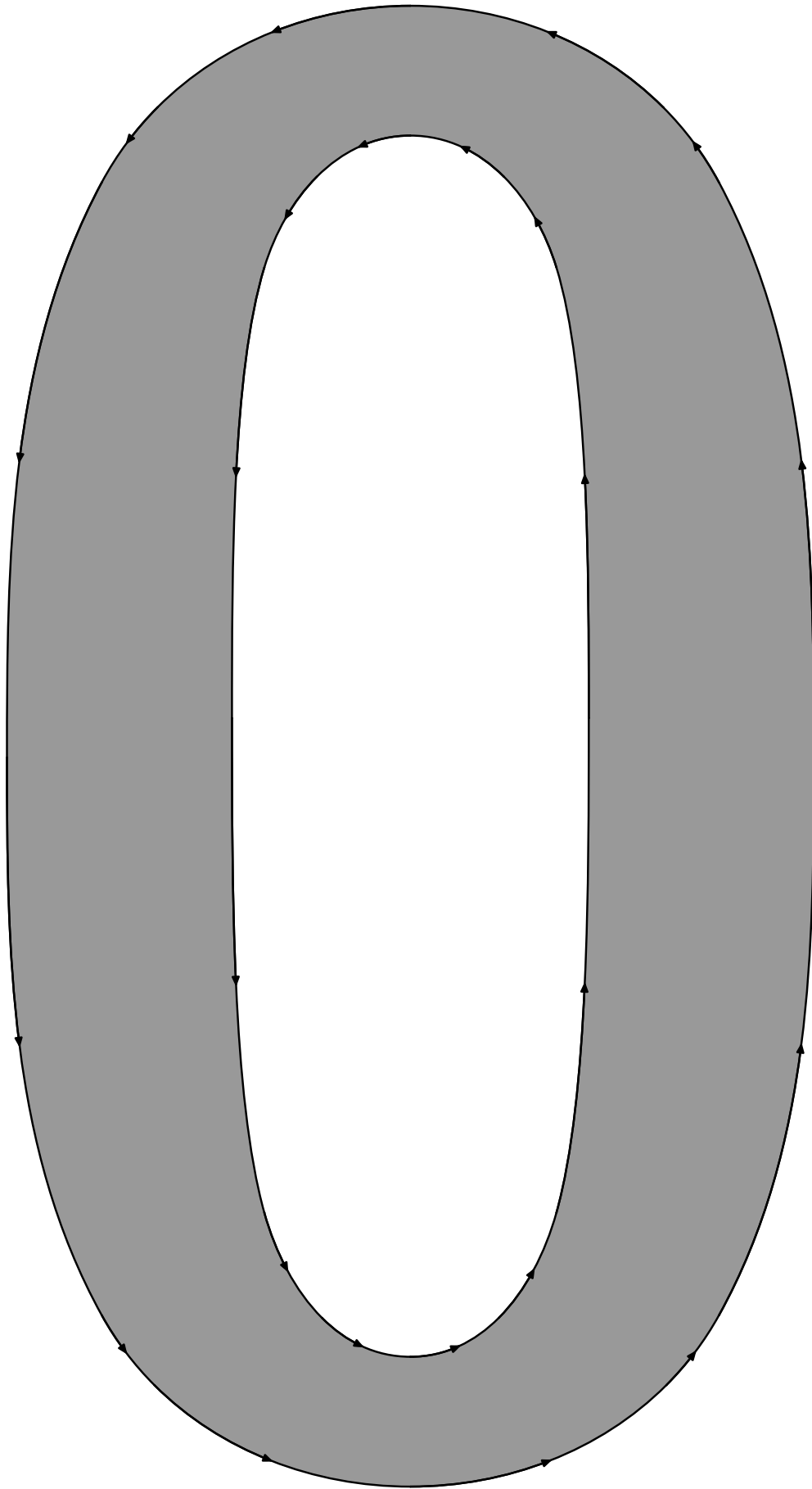


FIGURE 2: The glyph of the numeral 0 in `xmssdc10.mf` font.

12 for the inner one) but it should be clear that the METAFONT description is much more straightforward, or, at least, “typographic”.

Things completely change when we consider the numeral 2:

```
cmchar "The numeral 2";
beginchar("2",9u#,fig_height#,0);
italcorr fig_height#*slant-.5u#;
adjust_fit(0,0);
numeric arm_thickness, hair_vair;
hair_vair=.25[vair,hair];
arm_thickness=
Vround(if hefty:slab+2stem_corr
      else:.4[stem,cap_stem] fi);
pickup crisp.nib;
pos7(arm_thickness,-90); pos8(hair,0);
bot y7r=0; lft x7=hround .9u; rt x8=hround(w-.9u);
y8=good.y(y7l+beak/2)+eps;
arm(7,8,a,.3beak_darkness,beak_jut);%arm and beak
pickup fine.nib; pos2(slab,90);
pos3(.4[curve,cap_curve],0);
top y2r=h+o; x2=.5(w-.5u);
rt x3r=hround(w-.9u); y3+.5vair=.75h;
if serifs:
  numeric bulb_diam;
  bulb_diam=hround(flare+2/3(cap_stem-stem));
  pos0(bulb_diam,180); pos1(cap_hair,180);
  lft x1r=hround .9u; y1-.5bulb_diam=2/3h;
  (x,y2l)=whatever[z1l,z2r];
  x2l:=x; bulb(2,1,0); % bulb and arc
else: x2l:=x2l-.25u; pos1(flare,angle(-9u,h));
  lft x1r=hround .75u; bot y1l=vround .7h;
  y1r:=good.y y1r+eps; x1l:=good.x x1l;
  filldraw stroke term.e(2,1,left,.9,4);
fi % terminal and arc
pos4(.25[hair_vair,cap_stem],0);
pos5(hair_vair,0);
pos6(hair_vair,0);
y5=arm_thickness; y4=.3[y5,y3];
top y6=min(y5,slab,top y7l);
lft x6l=crisp.lft x7;
z4l=whatever[z6l,(x3l,bot .58h)];
z5l=whatever[z6l,z4l];
erase fill z4l--
z6l--lft z6l--
(lft x6l,y4l)--cycle;%erase excess at left
filldraw stroke z2e{right}..tension
  atleast .9 and atleast 1
  ..z3e{down}..{z5e-z4e}z4e--z5e--z6e;%stroke
penlabels(0,1,2,3,4,5,6,7,8);
endchar;
```

As we can see in fig. 3, there are both a contour and envelopes of more than a pen; there are intersections between the contour the envelopes and the pens, and some curves are outside the glyph (some of these curves are used to delete unwanted black pixels). There are also some unexpected straight lines and small curves. The number of curves looks quite large, which is not what we desire as we want to obtain the outline depicted in fig. 4.

Unfortunately, things are even different and it's necessary to describe how METAFONT calculates pen envelopes to go on. This is explained in the book “METAFONT: The Program” (KNUTH, 1986a) at the “Polygonal pens” part, chapter 469, that we briefly quote with a slightly modified notation:

Given a convex polygon with vertices $\mathbf{w}_0, \mathbf{w}_1, \dots, \mathbf{w}_{n-1}, \mathbf{w}_n = \mathbf{w}_0$ in *counterclockwise* order ... (and a curve $\mathbf{B}(t)$) the envelope is obtained if we offset $\mathbf{B}(t)$ by $\mathbf{w}_k$ when the curve is travelling in a direction $\mathbf{B}'(t)$ lying between the directions $\mathbf{w}_k - \mathbf{w}_{k-1}$ and $\mathbf{w}_{k+1} - \mathbf{w}_k$. At times t when the curve direction $\mathbf{B}'(t)$ increases past $\mathbf{w}_{k+1} - \mathbf{w}_k$, we temporarily stop plotting the offset curve and we insert a straight line from $\mathbf{B}(t) + \mathbf{w}_k$ to $\mathbf{B}(t) + \mathbf{w}_{k+1}$; notice that this straight line is tangent to the to the offset curve. Similarly, when the curve direction decreases past $\mathbf{w}_k - \mathbf{w}_{k-1}$, we stop plotting and insert a straight line from $\mathbf{B}(t) + \mathbf{w}_k$ to $\mathbf{B}(t) + \mathbf{w}_{k+1}$; the latter line is actually a “retrograde” step which will not be part of the final envelope under the METAFONT's assumptions. The result of this construction is a continuous path that consist of alternating curves and straight line segments.

This explains why the number of the curves is large and why there are small curves, but says nothing about those circular curves that we can see in fig. 4: METAFONT indeed converts an elliptical pen into a polygonal one and then applies the algorithm. The conversion is accurate enough to guarantee that the envelope is correctly filled with the right pixels. This is a key point to understand: *METAFONT's main task is to produce the best bitmap of a glyph, not the best outline.*

The role of the sensors is to gather as much information as possible about pixels, contours, the polygonal version of the pens, envelopes and their straight lines and then store these information (basically the edge structure of the pixels and Bézier curves with an eventual offset) into appropriate Lua tables. As METAFONT halts, the Lua interpreter calls `end_program.lua` and let the programmer manage these tables: sometimes, as we have seen in the numeral 0 case, the post-process can be quite simple, sometimes not. MFLua doesn't automatically output a glyph outline because it's the programmer who has to implement the best strategy according to his experience.

4 Two different strategies for post-processing the curves

4.1 The Concrete Roman 10 pt

The first use of MFLua has been the post-processing of Concrete Roman 10 pt to obtain an OpenType version of it. This font is described in the file `ccr10.mf`. As we previously said, sensors collect the data into Lua tables and `end_program.lua` post-processes them at the end of the execution (we could even choose to ex-

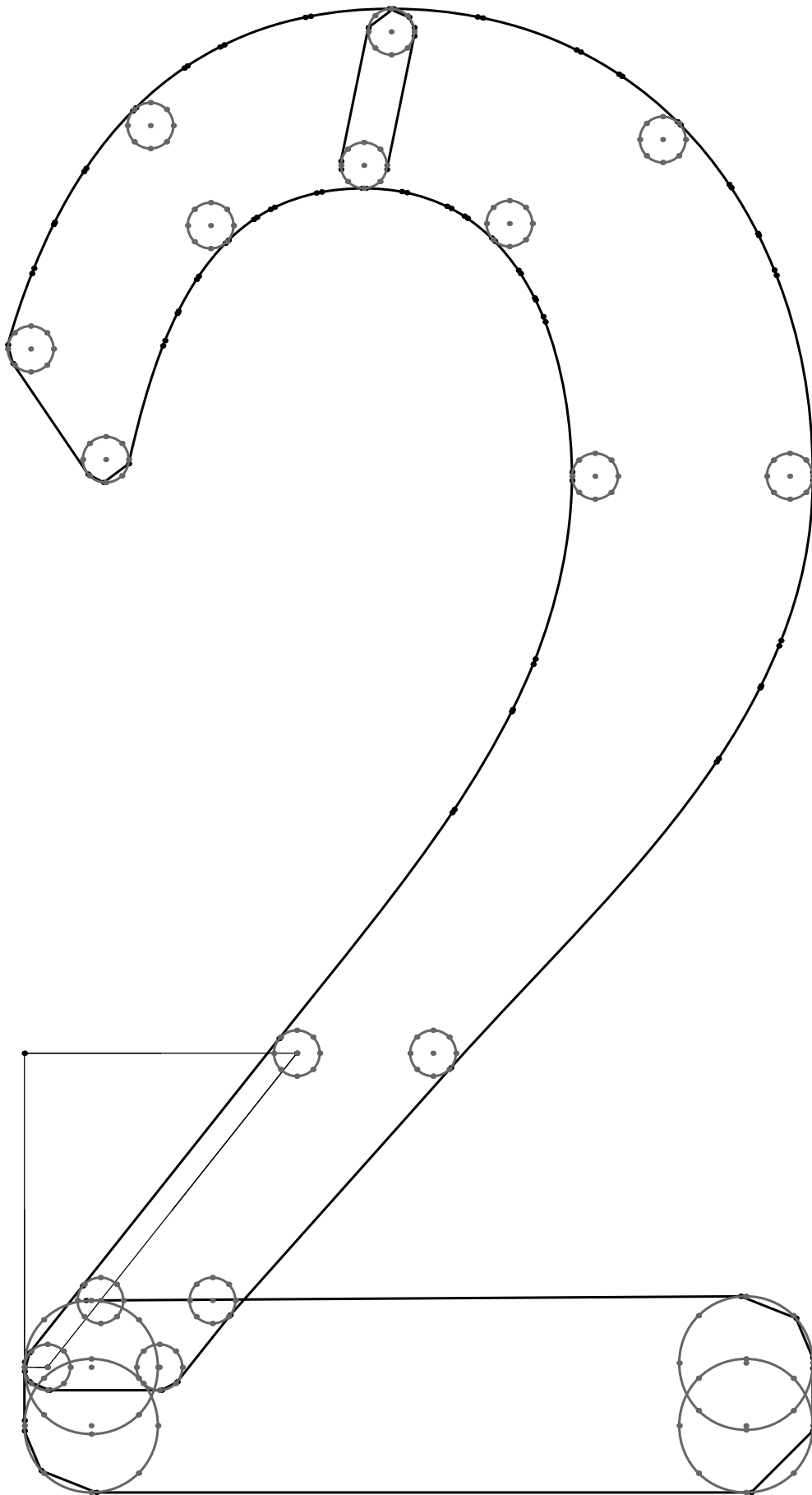


FIGURE 3: The glyph of the numeral 2 in `xmsdc10` font. We can see envelopes and pens (thick curves) and a contour (thin curve).

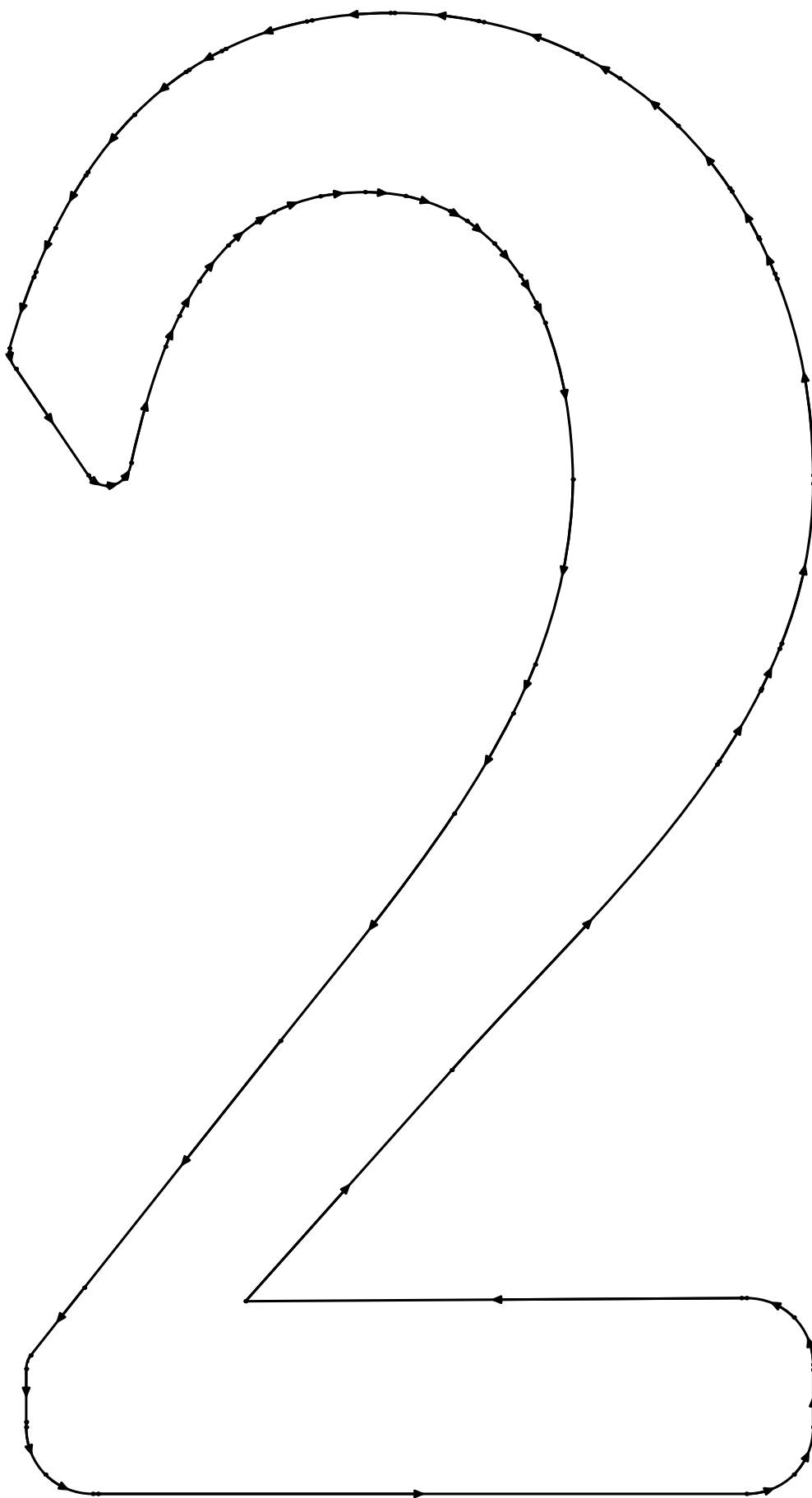


FIGURE 4: An outline of the numeral 2 in xmsdc10.mf font.

ecute the no-more post-process during the execution). The script `end_program.lua` defines the global array `chartable[index]` that contains the data for the glyph with char code `index`: we have the edge structure that allows the program to calculate the pixels of the glyph as well as the three arrays `valid_curves_c`, `valid_curves_e` and `valid_curves_p` that gather the data of contours, envelopes and the polygonal version of the pens. Each array contains the array of the control points $\{p, c_1, c_2, q\}$ stored as a string " $\langle x \rangle, \langle y \rangle$ ", where $\langle x \rangle$ and $\langle y \rangle$ are the coordinates of the point. With fig. 3 as a reference, we can see that when we draw a glyph with a pen it usually has overlapping strokes. Along with the curves of the pen(s), these overlaps create curves inside or outside the glyph that must be deleted. Having the pixels of the glyph, we can use the parametric form (1) to check if a point (x, y) (or better, a neighborhood with center (x, y)) is inside or outside. If all the points of the curve are inside or outside, we can delete them. The drawback is that while time t goes linearly in $\mathbf{B}(t)$, the points $(x(t), y(t))$ follow a cubic (i.e., not linear) law in case the curve is not a straight line. Hence, they are not equally spaced — this means that we can jump over some critical points. Using the same time interval steps for each curve means that short curves are evaluated in times where the points can differ less than a pixel — a useless evaluation. Of course not all the curves are inside the glyph: there are curves on the border and curves partially lying on the border and partially inside (or outside). In the latter case the result of evaluation is an array of time intervals where the curve crosses the border.

Once we have deleted the curves that are completely inside (or outside) the glyph, the next step is to merge all the curves and split them using the previously seen time interval (this is done by a Lua implementation of the De Casteljau's algorithm). Now we have a set of curves that are on the border or "near" it (i.e., partially on the border). We can delete those curves having only one intersection (a *pending curve*), supposing that each curve of the final outline has exactly 2 intersections at times $t_0 = 0$ and $t_1 = 1$.

To calculate all the intersections we use the following trick: if we have n curves, we produce a METAFONT file that contains the code that calculates the intersection between p_i and p_j for $1 \leq j \leq n$ and $j < i \leq n$ (given that $p_j \cap p_i = p_i \cap p_j$) and then we parse the log file with Lua. For example if

```
p1={"(57.401,351.877)",
      "(57.401,351.877)",
      "(57.901,349.877)",
      "(57.901,349.877)"}
and
```

```
p2={"(56.834,356.5)",
```

```
"(56.834,354.905)",
"(57.031,353.356)",
"(57.401,351.877)"}
then we have
```

```
batchmode;
message "BEGIN i=2,j=1";
path p[];
p1:=(57.401,351.877) ..
      controls (57.401,351.877) and (57.901,349.877) ..
      (57.901,349.877);
p2:=(56.834,356.5) ..
      controls (56.834,354.905) and (57.031,353.356) ..
      (57.401,351.877);
numeric t,u;
(t,u) = p1 intersectiontimes p2;
show t,u;
message "" ;
```

and the log

```
BEGIN i=2,j=1
>> 0
>> 0.99998
```

If the result is $(-1, -1)$ the intersection is empty. There are two problems with this approach: the first one shows when a curve crosses the border and time intervals can generate two curves, one completely outside and one completely inside — hence deleting an intersection. To avoid this issue we must adjust the intervals moving the extremes a bit. We have the second problem when there can be curves with three or more intersections — i.e., we can have *loops*. Opening a loop can be a difficult task: e.g., if the curve p_a intersects $\{p_b, p_c, p_d\}$ at the same time t_a and p_b intersects $\{p_a, p_c, p_d\}$ at t_b then $I_a = \{p_a\} \cup \{p_b, p_c, p_d\}$ is equal to $I_b = \{p_b\} \cup \{p_a, p_c, p_d\}$ and we can delete p_a and p_b because p_c and p_d stay connected. But with more than three intersections things become more complex.

To solve these cases, `end_program.lua` has a series of *filters*. A filter acts on a specific glyph and typically removes unwanted curves and/or adjusts the control points to ensure that a curve joins properly with its predecessors and successor. Of course this means that the programmer inspects each glyph separately, which is reasonable when we are designing the font — less reasonable when we convert it.

We can call this approach *per-font and per-glyph*: `end_program.lua` is a Lua script valid only for a specific font and which has filters for each glyph.

The script `end_program.lua` also has some functions to convert the outlines (with the correct turning number) of each glyph into a SVG font: this font format can be imported into FontForge and, usually after re-editing the glyphs (tipically simplifying the curves), it can be saved as an OpenType CFF. In fig. 5 we can see an example of this font.

4.2 The Computer Modern Sans Serif Demibold Condensed 10 pt

We now approach a more "geometric" strategy. We don't want to output an OpenType font but

ff fi fl ffi ffl Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam nisl urna, eleifend vel mollis quis, facilisis vel dolor. Sed auctor nibh eu magna vulputate vulputate. Curabitur ante mauris, pretium eu laoreet at, venenatis et neque. Vestibulum ante quam, tristique in posuere eu, pulvinar vel neque. Nam faucibus, neque ut commodo luctus, lacus risus accumsan felis, a feugiat lorem justo venenatis dui. Aenean bibendum tincidunt enim ac cursus. Vivamus a arcu a augue auctor consectetur nec sed augue. Quisque dignissim felis imperdiet mi lacinia suscipit. Maecenas nunc tortor, congue nec posuere sit amet, ultricies vel diam. In aliquam arcu eu lacus congue eget rutrum justo volutpat. Quisque ac nisi vitae leo fringilla lobortis.

Curabitur rhoncus lobortis ante, eget euismod magna blandit nec. Praesent non sem nulla. Sed congue magna sit amet libero sodales eu ultrices orci posuere. Suspendisse sed nibh a tortor fermentum ornare. Suspendisse vel felis eget tellus gravida rhoncus. Ut vel magna lacus, placerat semper enim. Vestibulum rutrum condimentum neque et adipiscing. Duis nulla enim, euismod a cursus id, ornare vel tellus. Vestibulum lobortis metus egestas velit euismod pellentesque. Praesent elit ante, consequat at posuere a, rhoncus id magna. Phasellus ut nisl orci, ac molestie eros. Suspendisse potenti. Suspendisse ac porttitor lorem. Curabitur eu elit sed neque placerat accumsan. Cras eu odio diam. Nunc lorem ligula, interdum eget consequat non, laoreet eget magna.

Maecenas consequat ultrices est, vitae rutrum nulla egestas sed. Proin rutrum lorem in sem posuere pretium. Cras accumsan euismod quam eget pulvinar. Maecenas eget posuere sem. Nulla sit amet luctus elit. Nulla vel ligula velit. Nunc consectetur orci a odio venenatis facilisis. Integer venenatis commodo nibh sed gravida. Ut ornare arcu in mi eleifend convallis. Quisque tincidunt, tellus et sodales interdum, nulla massa suscipit ante, non tincidunt ligula diam id nunc. In eu justo at lectus pulvinar accumsan. Vivamus convallis sodales ligula, ut gravida elit consectetur at. Ut in augue nec tortor vehicula vehicula eu eu lorem. Vivamus tristique neque ut tellus tristique aliquet.

Proin quis augue a elit convallis venenatis. Quisque scelerisque dictum augue condimentum rutrum. Integer nec dignissim nisl. Aenean vitae justo lectus, eu vulputate ipsum. Sed porttitor dapibus arcu sed faucibus. Sed vitae arcu eu quam ultrices ornare. In in est nec purus consequat vehicula. Integer ut fermentum dolor. Vivamus neque quam, cursus at viverra

FIGURE 5: The ConcreteOT font produced by MFLua from ccr10.mf.

to find an `end_program.lua` more *universal and per-glyph* and less *per-font and per-glyph*. Our experience with `ccr10.mf` make us believe that is always possible to write a METAFONT program that outputs a nice bitmap of a glyph using a very complex set of curves. This is especially true when we use pens and the need to manually correct every error arises. Up to now we only made few outlines of numerals.

There are new functions to trace a curve and to calculate the intersections between two cubics (both based on De Casteljau's bisection algorithm, an application of De Casteljau's algorithm) so the parametric form and the trick to calculate the intersections are not needed anymore. We also keep contours, envelopes and pens apart almost until the end of the process, when we first merge envelopes and pens and then, at last, contours. The most important enhancement is probably the replacement of the polygonal version of a pen with an elliptical one. METAFONT generates a polygonal getting an ideal ellipse with major axis, minor axis and the angle of rotation from the pen specifications and

then calls `make_ellipse`. Putting a sensor around helps us store the axis and theta into a Lua table, to be read later from `end_program.lua`. The next step is a trick again: we call MFLua with the following file:

```
batchmode;
fill fullcircle
  xscaled (majoraxis)
  yscaled (minoraxis)
  rotated (theta) shifted (0,0);
shipit;
bye.
```

where `majoraxis`, `minoraxis` and `theta` get the ellipse data. MFLua then saves the outlines of the filled ellipse into another file, from which they can be read by `end_program.lua`. This script then saves each elliptical pen in a table, with `p.c1..c2..q` as a key to be reused later, instead of the polygonal one. We can see the result in fig. 6: the approximation is quite good. This reduces the total number of curves and gives the glyph a more natural look.

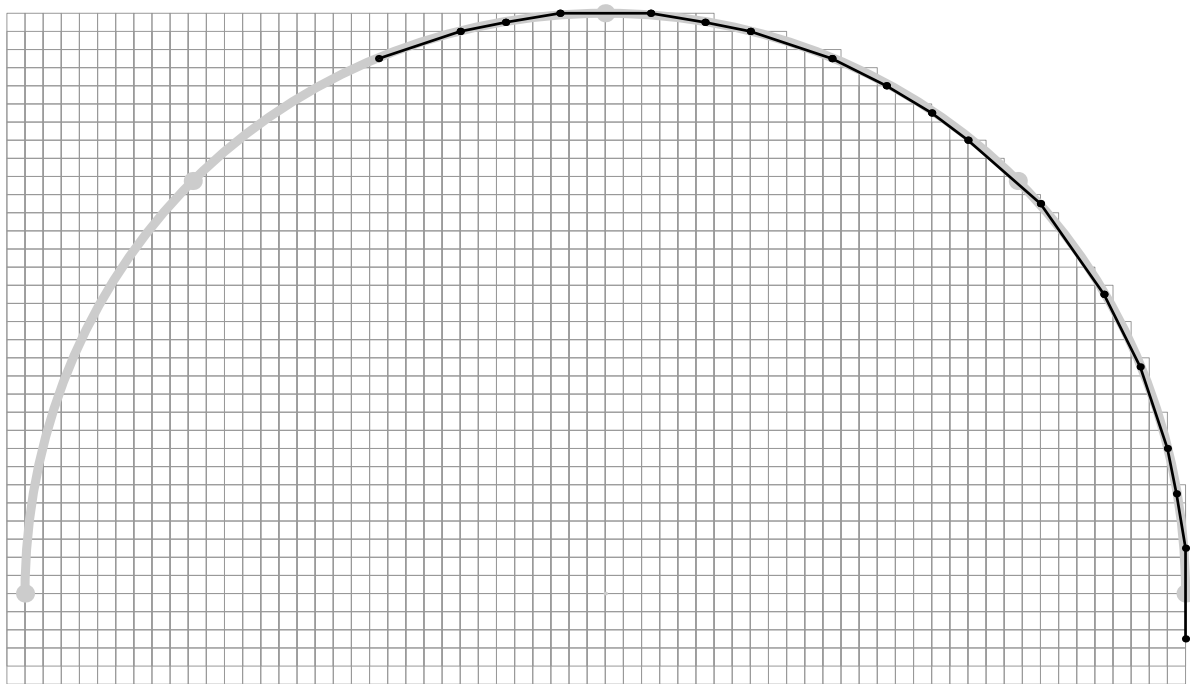


FIGURE 6: Real polygonal pen (black) vs. calculated elliptical pen (gray). Square boxes are the pixels. It's the bottom right part of fig. 3 .

5 Conclusion

We believe that MFLua is an interesting tool for font designers because too many fonts (if not all) are currently designed using contours. In this case `end_program.lua` should be simple (less or even no intersections, compared to the METAFONT technique, see the numeral 0 of `xmssdc10.mf`). On the other side, using the pens shows that extracting an outline is a difficult task. It's almost impossible to find an always valid script. The outlines from an envelope usually have a large number of curves, which is not a good feature, and this is a METAFONT property: we can always implement routines to simplify them, though FontForge already does it.

The work will continue on `xmssdc10.mf` to find an `end_program.lua` modular and flexible enough for a wide application.

References

- KNUTH, D. E. (1986a). *METAFONT: The Program*. Addison-Wesley, Massachusetts, 1^a edizione.
- (1986b). *The METAFONTbook*. Addison-Wesley, Massachusetts, 1^a edizione. — with final correction made in 1995 —.
- MARSH, D. (2005). *Applied Geometry for Computer Graphics and CAD*. Springer, London, 2^a edizione.

▷ Luigi Scarso
luigi dot scarso at gmail dot
com

Gestione ‘quasi automatica’ dei valori numerici in esempi di calcolo ed esercizi svolti

Agostino De Marco

Sommario

Nell'articolo viene discussa una tecnica di gestione dei valori numerici, utile specialmente alla preparazione di esempi di calcolo e di esercizi svolti. Questa è una necessità particolarmente sentita dagli autori di manuali tecnico-scientifici. I testi scientifici spesso contengono una successione di esempi di calcolo a corredo della parte teorica. Nella fase di stesura degli esercizi è bene mettere in pratica delle tecniche che limitano l'occorrenza di errori tipografici o addirittura concettuali. Si mostra qui una strategia in cui i dati numerici sono prodotti con strumenti esterni al sistema T_EX. Qualche cenno sarà dato a tecniche simili facenti uso di LuaL^AT_EX o del pacchetto di estensione l3fp.

Abstract

In this paper I will show some techniques to manage numbers in order to facilitate the preparation of calculation examples or worked out exercises. This is a well known problem for authors of scientific textbooks. When they want to include a gallery of examples with calculations in their book it is of utmost importance to adopt a strategy designed to limit both typographic and conceptual errors. The techniques shown here handle numeric values which are produced outside the typesetting process with external software tools. Finally, similar techniques are mentioned that use LuaL^AT_EX or the package l3fp.

1 Dalla teoria alla pratica

Nei manuali tecnici l'esposizione degli argomenti teorici è spesso intervallata da una successione di esercizi. Il lettore percepisce gli esempi di calcolo come delle verifiche ricorrenti che portano sul piano pratico i concetti che man mano vengono esposti.

Tipicamente gli esempi sono segnalati da particolari accorgimenti tipografici affinché si possa più facilmente individuare il flusso dell'esposizione teorica in cui sono immersi. Se ben congegnati, gli esercizi svolti costituiscono un potente strumento didattico oltre che a dare una sensazione di maggiore vicinanza della teoria alla realtà. Eventualmente in qualche esempio si farà riferimento ad esercizi precedenti per evitare ridefinizioni e risparmiare linee testo.

Prendiamo il caso di un autore che è nella fase di perfezionamento del suo testo di Aerodinamica.

Per terminare il manoscritto egli deve ultimare la parte esercitativa e deve quindi preparare una collezione di esempi di calcolo che andrà inserita nel testo.

Immaginiamo che a un certo punto del suo manuale il nostro autore presenti al lettore il concetto di “coefficiente di portanza” dicendo:

Il coefficiente di portanza C_L di un aeromobile che vola alla velocità V si definisce come

$$C_L = \frac{L}{\frac{1}{2}\rho V^2 S} \quad (1)$$

dove L è la portanza aerodinamica (*lift*), ρ è la densità dell'aria alla quota di volo ed S la superficie alare.

L'introduzione del concetto di coefficiente di forza adimensionale si presta ad un esempio di calcolo basato sulla formula (1). Volendo fornire al lettore un esempio realistico l'autore sceglie un velivolo di riferimento, esegue una ricerca per trovarne le caratteristiche ed annota quelle che servono per l'esempio da proporre. In questo caso è necessario fornire un valore realistico di L e della velocità di volo, ad esempio la velocità di crociera, ipotizzare una quota di volo per assegnare un valore a ρ , e fornire un valore corretto di S .

Dopo aver annotato i valori numerici necessari, ecco come si presenterà l'esempio di calcolo:

Esempio 1



Calcoliamo il coefficiente di portanza di un velivolo Boeing 747-400 che procede alla velocità di crociera $V_{cr} = 900$ km/h, alla quota $h = 11000$ m. Si può assumere una massa $m = 350000$ kg ed una superficie di riferimento $S = 541,2$ m² (l'area della forma in pianta dell'ala).

Alla quota di volo stabilita, secondo il modello di atmosfera standard (ISA, *International Standard Atmosphere*), si ha una densità dell'aria $\rho = 0,364$ kg/m³.

Detta $g = 9,81$ m/s² l'accelerazione di gravità, si ipotizza la condizione

$$L = mg \quad (2)$$

cioè che la portanza uguaglia il peso del velivolo.

Sulla base dei dati su riportati si calcola il peso

$$\begin{aligned} W = mg &= 350000 \text{ kg} \cdot 9,81 \text{ m/s}^2 \\ &= \underline{3,43 \cdot 10^6 \text{ N}} \end{aligned} \quad (3)$$

ed una forza di riferimento data dal denominatore a secondo membro della (1)

$$F_r = 0,5 \cdot 0,364 \text{ kg/m}^3 \cdot \left(900 \text{ km/h} \frac{1000 \text{ m/km}}{3600 \text{ s/h}} \right)^2 \cdot 541,2 \text{ m}^2 = \underline{6,16 \cdot 10^6 \text{ N}} \quad (4)$$

Infine, tenendo conto della (3), il valore di C_L è dato dal rapporto

$$C_L = \frac{W}{F_r} = \frac{3,43 \cdot 10^6 \text{ N}}{6,16 \cdot 10^6 \text{ N}} = \underline{0,558} \quad (5)$$



Questo esempio non presenta particolari difficoltà di calcolo. Evidentemente esso è utile all’interno di un testo di Aerodinamica di base perché:

- fornisce un’idea dell’ordine di grandezza del C_L di un noto velivolo in crociera,
- evidenzia che una delle grandezze necessarie a ottenere il risultato finale dipende dalla quota (va effettuato un calcolo intermedio basato su uno dei valori dati, valutando una funzione definita da uno specifico modello matematico di atmosfera),
- invita il lettore a verificare delle semplici operazioni di moltiplicazione e divisione.

Per preparare l’esercizio l’autore potrà semplicemente usare carta e penna, annotando i risultati intermedi e il risultato finale, inserendo poi a mano i valori numerici nel sorgente L^AT_EX del manoscritto.

Tenere traccia di tutti gli esempi svolti è un impegno che cresce ovviamente con il numero di esercizi che l’autore intende inserire nel volume. Alcuni di essi potrebbero essere ben più complessi, altri potrebbero essere una variazione sul tema rispetto ad alcuni proposti in precedenza.

Pertanto, per manoscritti di grandi dimensioni sarebbe utile escogitare una strategia di lavoro in cui viene automatizzato il calcolo dei valori numerici nonché l’inserimento dei risultati nel sorgente L^AT_EX.

Tipicamente nella pratica quotidiana un autore utilizza uno o più programmi di calcolo scientifico (Octave, ScicosLab, Sagemath, Matlab, Mathematica, Mathcad, eccetera) e conosce almeno un linguaggio di programmazione (Python, C++, Fortran, Java, Lua, eccetera); tutti strumenti, questi, in grado di esportare dati e formattare stringe su file di testo. Molti di questi software sono esterni al sistema T_EX ma possono essere utilizzati per sostituire il metodo ‘carta e penna’ e generare valori numerici affidabili. I numeri così generati potrebbero essere conservati in appositi file di testo per essere letti in fase di composizione del documento.

Va osservato che esiste anche la possibilità di effettuare calcoli all’interno del processo di composizione, ad esempio usando LuaL^AT_EX oppure il pacchetto l3fp di L^AT_EX3. Vedremo più avanti come ciò è possibile. In questa parte l’attenzione è rivolta all’uso di strumenti esterni per effettuare calcoli.

2 Un ambiente per gli esempi

Come si imposta in L^AT_EX l’esempio di calcolo presentato sopra?

In questo caso particolare è stato utilizzato il pacchetto ntheorem per gestire la numerazione degli esempi. Inoltre è stato creato un nuovo ambiente myExample. Ecco il codice da inserire nel preambolo:

```
\usepackage{relsize} % font size change
\usepackage{pifont} % for dingbats
\usepackage{marvosym} % for \Keyboard etc
\usepackage{mathtools} % for amsmath
\usepackage{adjustbox} % advanced boxes
\usepackage[amsmath,hyperref]{ntheorem}
% new theorem-like environment
\newtheorem{myExampleT}{}
% myExample label & format
\newcommand\myExampleLabel{Esempio}
\newcommand\myExampleLabelFormat{%
  \textbf{\upshape\hspace{3pt}\myExampleLabel}%
  \ \themyExampleT}%
}
\newcommand\myExampleMarkPencilKeyboardMouse{%
  \bf\ding{46}\ %
  \raisebox{-2pt}[0pt][0pt]{%
    \relsize{4}\Keyboard\hspace{2pt}%
    \ComputerMouse}%
}
}
\newcommand\myExampleMarkKeyboardMouse{%
  \raisebox{-2pt}[0pt][0pt]{%
    \relsize{4}\Keyboard\hspace{2pt}%
    \ComputerMouse}%
}
}
\newcommand\myExampleEndMark{%
  {\ding{118}}% \ding{111}
}
% environment per gli Esempi
\newenvironment{myExample}[1][\ding{46}]{%
  \begin{myExampleT}%
  \adjustbox{%
    set height=1.1\baselineskip,
    set depth=0.5\baselineskip, valign=m,
    center=\linewidth, bgcolor=gray!15}{%
    \adjustbox{left=0.6\linewidth}{%
      \myExampleLabelFormat%
    }%
    \adjustbox{right=0.4\linewidth}{%
      #1\ % myExample mark
    }
  }
}
\medskip
\par\upshape% testo normale
}%
\end{myExampleT}
\smallskip
\adjustbox{
  set height=0.55\baselineskip,
  set depth=0.11\baselineskip, valign=m,
  center=\linewidth, bgcolor=gray!15}{%
  \relsize{-2}\myExampleEndMark%
}%
```

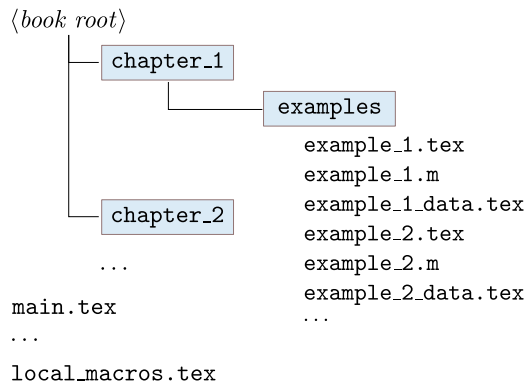


FIGURA 1: Organizzazione del materiale relativo agli esempi di calcolo in un manoscritto di grandi dimensioni.

```

\medskip
}

```

La macro `\newtheorem` imposta un nuovo tipo di teorema, chiamato `myExampleT`. La macro `\newenvironment` crea il nuovo ambiente `myExample`. L'ambiente accetta un argomento opzionale che permette all'utente di visualizzare un simbolo nella parte destra dell'intestazione. Il simbolo predefinito è `\ding{46}` (☞) del pacchetto `pifont`. Per esercizi che richiedono di impostare programmi di calcolo o comunque di lavorare al computer si potranno utilizzare i simboli `\Keyboard` (☞) e `\ComputerMouse` (☞) del pacchetto `marvosym`.

L'ambiente `myExample` viene utilizzato all'interno di un documento $\text{\LaTeX}$ come segue:

```

\begin{myExample}
\noindent%
Calcoliamo il coefficiente di portanza di un
velivolo Boeing-747-400 in volo alla velocità
di crociera ...
\end{myExample}

```

Il comando di apertura `\begin{myExample}` dell'ambiente per l'esempio apre a sua volta un teorema `myExampleT` ed imposta tipograficamente l'intestazione (posizionando in una fascia grigia larga `\linewidth` l'intestazione "Esempio 1" e il simbolo della matita giustificato a destra). Il comando di chiusura `\end{myExample}` chiude anche il teorema ed imposta tipograficamente la fascia grigia di chiusura (con il simbolo $\diamond$ al centro).

3 Valori numerici importati da file

Come si può rendere più raffinata l'infrastruttura del codice $\text{\LaTeX}$ che genera l'esempio di calcolo?

La figura 1 suggerisce una possibile organizzazione dei file e delle cartelle di un manoscritto di grandi dimensioni. Tutto il materiale si trova nella cartella `<book root>`. Il file `main.tex` è il sorgente principale da compilare con `pdflatex` (o con le varianti `xelatex` o `lualatex`). Il file `local_macros.tex` raccoglierà tutti i comandi personalizzati. Il materiale di ciascun capitolo sarà collocato in un'apposita cartella (`chapter_1`, `chapter_2`, eccetera). A loro

volta, i file relativi agli esempi del singolo capitolo potranno risiedere in sottocartelle `examples`.

Il testo dell'esempio 1 del capitolo 1 – la porzione di codice in cui viene usato l'ambiente `myExample` – sarà contenuto nel file `example_1.tex` e sarà inserito nel manoscritto con il comando `\input{chapter_1/examples/example_1}`.

Va ricordato che la fase di stesura del manoscritto e degli esercizi è sempre un processo iterativo, fatto di continui aggiustamenti, di revisioni drastiche di alcune parti, di correzioni dell'ultimo momento. Tra un'iterazione e l'altra si potrebbe presentare l'esigenza, ad esempio, di apportare semplicemente una correzione ai dati di un esercizio. Con la strategia della 'carta e penna' l'autore sarebbe costretto a svolgere ancora una volta i calcoli e a reinserire a mano i dati nel sorgente $\text{\LaTeX}$.

In questi casi è più efficiente predisporre per ciascun esercizio un piccolo programma di calcolo che salva in un file di testo i valori numerici dei dati e del risultato. Il file verrà sovrascritto ogni volta che si lancerà il programma di calcolo collegato all'esercizio in lavorazione.

3.1 Un automatismo basato su Matlab

Prendiamo il caso in cui l'autore preferisca lavorare con il software di calcolo scientifico Matlab¹. L'esercizio potrà essere impostato e risolto attraverso un apposito programma in linguaggio Matlab.

Dalla figura 1 si osserva la presenza dei due file `example_1.m` e `example_1_data.tex` nella cartella `examples`. È chiaro che essi sono associati all'esempio 1. Il file `example_1.m` è uno *script* in linguaggio Matlab (detto anche 'M-file') che crea automaticamente in output il file `example_1_data.tex`, un sorgente $\text{\LaTeX}$. Ecco il codice Matlab contenuto nel file `example_1.m`:

```

h = 11000; % m
% calculate: air temperature, sound speed,
% pressure, density from ISA model
% (aerospace toolbox needed)
[T, a, P, rho] = atmosisa(h);
m = 350000; % kg
V_cruise = ...
    convvel(900,'km/h','m/s'); % m/s
g = 9.81; % m/s^2
S = 541.2; % m^2
L = m*g
F_r = 0.5*rho*V_cruise^2*S
C_L = L/F_r

% write on file
fileName = 'example_1_data.tex';
fileID = fopen(fileName,'w');
fprintf(fileID,'% -- FILE: %s\n',fileName);
fprintf(fileID, ...
    '\def\myAltitudeMT{%g}\n',h);
fprintf(fileID, ...
    '\def\myMassKG{%g}\n',m);
fprintf(fileID, ...
    '\def\myCruiseSpeedMTS{%g}\n',V_cruise);
fprintf(fileID, ...
    '\def\myCruiseSpeedKMH{%g}\n', ...

```

1. <http://www.mathworks.com>

```
convvel(V_cruise,'m/s','km/h'));
fprintf(fileID, ...
'\def\myGravityAccelerationMTSsquared{%g}\n',g);
fprintf(fileID, ...
'\def\myWingSurfaceMTSsquared{%g}\n',S);
fprintf(fileID, ...
'\def\myAirDensityKGMTcubed{%g}\n',rho);
fprintf(fileID, ...
'\def\myReferenceForceN{%g}\n',F_r);
fprintf(fileID, ...
'\def\myLiftN{%g}\n',L);
fprintf(fileID, ...
'\def\myCL{%g}\n',C_L);
fclose(fileID);
```

Questo programma è molto semplice:

- assegna i valori dei dati ad opportune variabili,
- invoca una funzione (`atmosisa`) per il calcolo della densità dell’aria alla quota assegnata,
- procede al semplice calcolo del risultato finale,
- infine crea un file di testo e scrive delle stringhe attraverso la funzione `fprintf`.

Lo script deve essere eseguito all’interno dell’ambiente Matlab nella cartella di lavoro `examples`. Esso crea il file nominato `example_1_data.tex` che si presenta così:

```
% -- FILE: example_1_data.tex
\def\myAltitudeMT{11000}
\def\myMassKG{350000}
\def\myCruiseSpeedMTS{250}
\def\myCruiseSpeedKMH{900}
\def\myGravityAccelerationMTSsquared{9.81}
\def\myWingSurfaceMTSsquared{541.2}
\def\myAirDensityKGMTcubed{0.363918}
\def\myReferenceForceN{6.15476e+06}
\def\myLiftN{3.4335e+06}
\def\myCL{0.55786}
```

Questo sorgente, definisce delle macro personalizzate attraverso il comando primitivo `\def` di T_EX. Ciascuna macro contiene un valore numerico da inserire nel testo dell’esercizio. Il sorgente deve essere caricato dal file `example_1.tex` con il comando `\input{chapter_1/examples/example_1_data}`. Quest’ultimo permetterà di chiudere la catena di gestione ‘quasi automatica’ dei dati.

Ad esempio, nel sorgente `example_1.tex`, a valle delle definizioni delle macro personalizzate, un comando come `\myAirDensityKGMTcubed` sarà espanso da T_EX ottenendo la sequenza `0.363918`. In particolare si potrà usare il comando `\SI` del pacchetto `siunitx` per comporre correttamente un’espressione come quella seguente:

$$\rho = 0,364 \text{ kg/m}^3$$

ottenuta con il codice:

```
%----- nel preambolo
\usepackage{siunitx}
\sisetup{
load=derived, unitsep=thin,
valuesep=thin, decimalsymbol=comma,
round-mode=places,
expproduct=cdot,
group-separator={\thinspace},
sepfour=false
}
%-----
```

```
...
\[
\rho =
\SI[round-precision=3]
{\myAirDensityKGMTcubed}{kg/m^3}
\]
```

La figura 2 mostra il codice completo del file `example_1.tex` che compone il testo dell’esempio 1. Scorrendo il codice L^AT_EX si osserva che l’esempio di calcolo è stato completamente ‘parametrizzato’. I parametri sono rappresentati dalle macro che contengono i dati del problema: `\myCruiseSpeedKMH`, `\myAltitudeMT`, `\myMassKG`, `\myWingSurfaceMTSsquared`; poi ci incontrano le macro contenenti i risultati intermedi: `\myAirDensityKGMTcubed`, `\myLiftN`, `\myReferenceForceN`; infine la macro con il risultato finale: `\myCL`.

Con l’M-file `example_1.m` che ha per output il file `example_1_data.tex` i valori numerici vengono gestiti esternamente al processo di composizione. Il problema viene risolto con un software scientifico e i risultati vengono opportunamente assegnati a delle macro in un file apposito. All’autore è sufficiente richiamare il file che assegna le macro con un comando `\input` immediatamente prima di aprire l’ambiente `myExample`.

L’automatismo proposto qui *non* è perfetto perché richiede all’utente di intervenire lanciando il programma `example_1.m`.² Tuttavia questa operazione è molto più comoda del completo rifacimento a mano dei calcoli dell’esercizio proposto. Inoltre l’infrastruttura della figura 1 richiede la buona pratica di tenere ben in ordine l’archivio degli esercizi a corredo del testo. Infine, la necessità di lavorare con un’applicazione scientifica in parallelo al sistema T_EX non dovrebbe essere psicologicamente ingombrante per l’autore scientifico, abituato per il suo lavoro quotidiano a interfacciarsi con una o più applicazioni di calcolo.

Questa tecnica di preparazione degli esercizi svolti è ugualmente valida per un qualsiasi altro strumento di calcolo esterno. Ciò che è richiesto al software scelto è che esso sia in grado di creare un file come `example_1_data.tex`.

3.2 Un automatismo basato su Mathcad

Nel caso precedente il software scientifico era in grado di creare direttamente un sorgente L^AT_EX grazie alla possibilità offerta dalla funzione `fprintf` (ereditata dal linguaggio C) di formattare stringhe. Vediamo qui invece un caso in cui il software scelto

2. In realtà all’interno del codice L^AT_EX si può dare il comando `\write18`, che chiede al sistema di eseguire l’argomento di questa macro, come se fosse un comando scritto nella console del sistema operativo (BECCARI, 2012). In questo caso è possibile chiedere di far eseguire a Matlab un determinato M-file con un comando del tipo:

```
matlab -nodisplay -nosplash -nodesktop \
-r "run('path-to-M-file)/(M-file).m');"
```

```

\input{chapter_1/examples/example_1_data}
\begin{myExample}
\noindent%
Calcoliamo il coefficiente di portanza di un velivolo Boeing-747-400 in volo alla velocità di crociera
 $V_{\text{cr}} = \text{SI}\{\text{myCruiseSpeedKMH}\}\{\text{km/h}\}$ , alla quota  $h = \text{SI}\{\text{myAltitudeMT}\}\{\text{m}\}$ .
Si può assumere una massa  $m = \text{SI}\{\text{myMassKG}\}\{\text{kg}\}$  ed una superficie di riferimento
 $S = \text{SI}\{\text{myWingSurfaceMTsquared}\}\{\text{m}^2\}$  (l'area della forma in pianta dell'ala).

Alla quota di volo stabilita, secondo il modello di atmosfera standard
(ISA, \emph{International Standard Atmosphere}), si ha una densità dell'aria
 $\rho = \text{SI}\{\text{myAirDensityKGMTcubed}\}\{\text{kg/m}^3\}$ .

Detta  $g = \text{SI}\{\text{myGravityAccelerationMTsquared}\}\{\text{m/s}^2\}$  l'accelerazione di gravità, si ipotizza
la condizione
\begin{equation}\label{eq:Lift:equal:Weight}
L = mg
\end{equation}
cioè che la portanza uguaglia il peso del velivolo.

Sulla base dei dati su riportati si calcola il peso
\begin{equation}\label{eq:Weight:calc}
\begin{split}
W = mg = \text{SI}\{\text{myMassKG}\}\{\text{kg}\} \cdot \text{SI}\{\text{myGravityAccelerationMTsquared}\}\{\text{m/s}^2\} \\
= \underline{\text{SI}\{\text{myLiftN}\}\{\text{N}\}}
\end{split}
\end{equation}
ed una forza di riferimento data dal numeratore a secondo membro della (\ref{eq:CL:Definition}):
\begin{equation}\label{eq:Fr:calc}
\begin{split}
F_{\text{r}} = \text{SI}\{\text{myAirDensityKGMTcubed}\}\{\text{kg/m}^3\} \cdot \left( \frac{\text{SI}\{\text{myCruiseSpeedKMH}\}\{\text{km/h}\}}{\text{SI}\{3600\}\{\text{s}\}} \right)^2 \cdot \text{SI}\{\text{myWingSurfaceMTsquared}\}\{\text{m}^2\} \\
= \underline{\text{SI}\{\text{myReferenceForceN}\}\{\text{N}\}}
\end{split}
\end{equation}
Infine, il valore di  $C_L$  è dato dal rapporto
\begin{equation}\label{eq:CL:calc}
C_L = \frac{W}{F_{\text{r}}} = \frac{\text{SI}\{\text{myLiftN}\}\{\text{N}\}}{\underline{\text{SI}\{\text{myReferenceForceN}\}\{\text{N}\}}}
\end{equation}
\end{myExample}

```

FIGURA 2: Codice L^AT_EX dell'esempio 1 (file `example_1.tex`) che utilizza i dati prodotti dallo script Matlab `example_1.m`.

dall'autore è Mathcad Prime³, un'applicazione per il calcolo scientifico molto potente ma non molto flessibile in termini di esportazione dei dati in formato testuale.

L'aspetto chiave della tecnica di gestione dei dati che si vuole illustrare non è diverso da quello del caso precedente: creare un file di testo in cui scrivere i valori numerici e i nomi delle macro da usare come parametri nel sorgente L^AT_EX. Per comporre l'esempio di calcolo si continuerà a utilizzare il file sorgente `example_1.tex` della figura 2, sostituendo il comando `\input` che precede l'apertura dell'ambiente `myExample` con la chiamata ad una macro diversa.

La figura 3 mostra un foglio di calcolo Mathcad Prime (versione 2) con il quale è stato risolto il problema proposto nell'esempio 1.

Mathcad Prime (così come il suo predecessore Mathcad 15) è noto per essere un software scientifico molto intuitivo e potente allo stesso

tempo. L'utente percepisce il foglio di lavoro come una pagina bianca sulla quale può scrivere testo, definire e assegnare variabili usando i comuni simboli matematici, definire funzioni, impostare calcoli simbolici, disegnare grafici. Oltre alle funzionalità di calcolo è possibile accedere a strumenti di programmazione e importare altri fogli di calcolo, con grandi potenzialità di riuso del codice. Tutte queste operazioni vengono svolte in modo naturale con l'aiuto di un caratteristico *ribbon* degli strumenti ispirato a quello di Microsoft Office 2010, posto nella parte superiore dello schermo.

Tra le caratteristiche notevoli di Mathcad vi è la gestione intelligente delle unità di misura associate alle variabili. Definita una variabile dotata di unità di misura, è subito possibile conoscerne il valore espresso in un sistema di misura diverso. L'unità di misura associata ad operazioni tra grandezze fisiche è gestita automaticamente. Il software presenta anche un gran numero di costanti fisiche predefinite.

3. <http://www.ptc.com/product/mathcad/>

PRELIMINARY OPERATIONS $mOut1 := [999 \text{ "myInitialValueA"}]$ Initialize output matrix

AIRCRAFT GENERAL DATA (Boeing 747-400)

$m_{TOT} := 350000 \text{ kg} = (7.716 \cdot 10^5) \text{ lb}$ Aircraft mass

$W_{TOT} := m_{TOT} \cdot g$ Aircraft weight $W_{TOT} = (3.5 \cdot 10^5) \text{ kgf}$ $W_{TOT} = (3.432 \cdot 10^6) \text{ N}$

$h_{ASL} := 11000 \text{ m}$ Above-Sea-Level (ASL) flight altitude $h_{ASL} = (3.609 \cdot 10^4) \text{ ft}$

AIR DENSITY CALCULATION ISA atmospher model

$R_{air} := 287 \cdot \frac{\text{N} \cdot \text{m}}{\text{kg} \cdot \text{K}}$ Perfect gas constant of air

$\gamma_{air} := 1.4$ Air adiabatic index (specific heat coefficient ratio)

$LR_{ISA} := -0.0065 \cdot \frac{\text{K}}{\text{m}}$ Lapse rate (LR), valid for troposphere only, up to 36000 ft

$T_{SL} := 288.16 \cdot \text{K}$ Temperature at sea level (SL)

$f_{T_{ISA}}(h) := T_{SL} + LR_{ISA} \cdot h$ Air temperature at altitude h

$f_{a_{ISA}}(h) := \sqrt{\gamma_{air} \cdot R_{air} \cdot f_{T_{ISA}}(h)}$ Air sound speed at altitude h

$\rho_{SL} := 1.225 \cdot \frac{\text{kg}}{\text{m}^3}$ Air density at sea level

$f_{\sigma_{ISA}}(h) := \left(\frac{f_{T_{ISA}}(h)}{T_{SL}} \right)^{-\left(\frac{g}{LR_{ISA} \cdot R_{air}} + 1 \right)}$ Density ratio at altitude h

$f_{\rho_{ISA}}(h) := \rho_{SL} \cdot f_{\sigma_{ISA}}(h)$ Air density at altitude h

Power term:
 $-\left(\frac{g}{LR_{ISA} \cdot R_{air}} + 1 \right) = 4.257$

Check results
 $\rho := f_{\rho_{ISA}}(h_{ASL})$ $\rho = 0.364 \frac{\text{kg}}{\text{m}^3}$ $a := f_{a_{ISA}}(h_{ASL}) = 295.049 \frac{\text{m}}{\text{s}}$

FLIGHT SPEED $V_{cr} := 900 \text{ kph}$ $Mach := \frac{V_{cr}}{a} = 0.847$ $V_{cr} = 250 \frac{\text{m}}{\text{s}}$

WING DATA $S_W := 541.2 \text{ m}^2$

REFERENCE FORCE $F_r := \frac{1}{2} \cdot \rho \cdot V_{cr}^2 \cdot S_W = (6.153 \cdot 10^6) \text{ N}$

LIFT COEFFICIENT $C_L := \frac{W_{TOT}}{F_r} = 0.558$

STACK DATA-MACRO COUPLES

$mOut1 := \text{stack}(mOut1, [\text{round}(m_{TOT} \cdot \text{kg}^{-1}, 1) \text{ "myMassKG"}])$

$mOut1 := \text{stack}(mOut1, [\text{round}(W_{TOT} \cdot \text{N}^{-1}, 1) \text{ "myLiftN"}])$

$mOut1 := \text{stack}(mOut1, [\text{round}(h_{ASL} \cdot \text{m}^{-1}, 1) \text{ "myAltitudeMT"}])$

$mOut1 := \text{stack}(mOut1, [\text{round}(C_L, 3) \text{ "myCL"}])$

WRITE OUTPUT FILES

$res := \text{WRITE_TEXT}(\text{"example_1_data.txt"}, mOut1)$

FIGURA 3: Foglio di calcolo Mathcad Prime 2 che genera un file di output testuale con i dati relativi all'esempio 1.

Con riferimento alla figura 3, si osservi, ad esempio, la variabile m_{TOT} , assegnata in kg con l'operatore $:=$. Successivamente il suo valore è visualizzato in lb con l'operatore $=$ (digitando l'unità desiderata). Il peso del velivolo è assegnato alla variabile W_{TOT} come prodotto " $m_{\text{TOT}} \cdot g$ ", dove g è la costante predefinita corrispondente all'accelerazione di gravità sulla superficie terrestre. L'operatore $=$ applicato successivamente a W_{TOT} ne visualizza il corretto valore in N e in kp (kilopond o kg-forza, "kgf").

Mathcad Prime offre una caratteristica che rendere possibile la manipolazione dei dati di output da parte di TEX: è consentito costruire matrici con elementi di tipo misto (numeri puri e stringhe). Una matrice di sole due colonne, di cui la prima è costituita da numeri e la seconda da stringhe con nomi di macro TEX, può essere esportata in un file di testo con la funzione predefinita "WRITETEXT". Nel foglio di calcolo della figura 3 l'ultima istruzione esegue la suddetta operazione invocando la funzione di scrittura per un file `example_1_data.txt` contenente la matrice m_{Out1} .

La matrice che fa da contenitore delle informazioni da esportare viene inizializzata con la prima istruzione in cima al foglio di lavoro:

$$m_{\text{Out1}} := \begin{bmatrix} 999 & \text{"myInitialValueA"} \end{bmatrix}$$

Essa ha dimensioni iniziali (1×2) e contiene i valori fittizi 999 e "myInitialValueA", cioè un numero al posto $(1, 1)$ e una stringa al posto $(1, 2)$.

Successivamente, man mano che le variabili sono assegnate o calcolate, la matrice viene aumentata aggiungendo nuove righe. Queste operazioni sono effettuate con la funzione predefinita "stack". Nella parte finale del foglio di calcolo si notano i comandi:

$$m_{\text{Out1}} := \text{stack} \left(m_{\text{Out1}}, \begin{bmatrix} \text{round}(m_{\text{TOT}} \cdot \text{kg}^{-1}, 1) & \text{"myMassKG"} \end{bmatrix} \right)$$

$$m_{\text{Out1}} := \text{stack} \left(m_{\text{Out1}}, \begin{bmatrix} \text{round}(W_{\text{TOT}} \cdot \text{N}^{-1}, 1) & \text{"myLiftN"} \\ \dots \end{bmatrix} \right)$$

$$m_{\text{Out1}} := \text{stack} \left(m_{\text{Out1}}, \begin{bmatrix} \text{round}(C_L, 3) & \text{"myCL"} \end{bmatrix} \right)$$

che ridefiniscono m_{Out1} accodandovi una matrice (1×2) di tipo $[\langle \text{numero} \rangle \langle \text{stringa} \rangle]$. Si osservi l'uso della funzione predefinita "round" che arrotonda il valore degli elementi in prima colonna con un numero di cifre decimali stabilite opportunamente. Inoltre si osservi che le variabili con unità di misura

devono essere necessariamente trasformate in numeri puri (lo richiede Mathcad); pertanto, variabili come m_{TOT} , W_{TOT} , eccetera, vengono divise per la corrispondente unità. Variabili adimensionali come C_L non hanno bisogno di questo accorgimento.

Si lascia al lettore l'approfondimento delle altre istruzioni contenute nel foglio di calcolo presentato nella figura 3. Per una guida su Mathcad Prime si veda WESSELINGH e DE WAARD (2012).

Il seguente listato mostra il contenuto del file di testo `example_1_data.txt` creato con la tecnica fin qui discussa:

```
999____"myInitialValueA"
350000____"myMassKG"
3432327.5____"myLiftN"
11000____"myAltitudeMT"
287____"myISAAirGasConstNMTKGK"
1.4____"myISAAirAdiabaticIndex"
-0.0065____"myISALapseRateKMT"
288.16____"myISAAirTemperatureSeaLevelK"
1.225____"myISAAirDensitySeaLevelKGMTcube"
0.364____"myAirDensityKGMTcube"
0.847____"myCruiseMach"
900____"myCruiseSpeedKMH"
250____"myCruiseSpeedMS"
9.81____"myGravityAccelerationMTSsquared"
541.2____"myWingSurfaceMTSsquared"
6153357.817____"myReferenceForceN"
0.558____"myCL"
```

In ciascuna riga del file comparirà un numero in virgola fissa seguito da un carattere di tabulazione e da una stringa delimitata da una coppia di doppi apici (").

In questo caso il problema di automatizzazione che si presenta all'autore è più delicato di quello risolto agevolmente in precedenza con Matlab. La versione attuale di Mathcad Prime non permette una formattazione dell'output testuale più conveniente di quella appena mostrata.

Dal file di output, che già contiene le informazioni giuste per lo scopo prefissato – cioè i numeri e i nomi delle macro che li conterranno (senza il primo carattere '\') – si vuole arrivare a definire una funzionalità che, ad esempio, prenda il numero 350000 e lo assegni ad una macro `\myMassKG`. Analogamente, si vuole iterare questa operazione per tutte le righe del file di testo. La soluzione del problema è ottenuta con TEX sfruttandone la possibilità di programmare l'apertura e la lettura dei file di testo e definire a basso livello nomi di macro con il comando `\@namedef`.

Ecco la porzione di codice, da inserire nel preambolo, che definisce una funzione di lettura dei file formattati come `example_1_data.txt`:

```
% see: http://tex.stackexchange.com/questions/47519
%% Parse Mathcad output
\makeatletter
\newread\myMathcadOutput
\def\parseline#1 "#2"{\@namedef{#2}{#1}}
\newcommand{\parseMathcadOutput}[1]{%
  \def\@tempb{\par}%
  \openin\myMathcadOutput=#1
  \loop\unless\ifeof\myMathcadOutput
    \read\myMathcadOutput to \@tempa
    \ifx\@tempa\@tempb\else
```

Esame di Meccanica del volo — Modulo di Manovre e Stabilità — Prova scritta dell'8 febbraio 2012

Sia dato un velivolo bimotore, propulso a elica, in volo alla quota del mare, caratterizzato dai dati riportati nelle tabelle seguenti.

Tabella 1 Dati globali del velivolo.

Massa totale, $m = W/g$	2500 kg
Coefficiente di resistenza a portanza nulla, C_{D0}	0,030
Fattore di Oswald della polare, e_{tot}	0,79
Posizione adimensionale del baricentro rispetto al bordo d'attacco della corda media aerodinamica, $X_{cg}/\bar{c}$	0,280

Tabella 2 Dati della fusoliera.

Coefficiente di momento di beccheggio a portanza nulla, $C_{M_{0,f}}$	-0,059
Gradiente del coefficiente di momento di beccheggio, $C_{M_{\alpha,f}}$	0,0062 deg ⁻¹
Gradiente del coefficiente di momento di imbardata, $C_{N_{\beta,f}}$	-0,00122 deg

Tabella 3 Dati caratteristici dell'ala. (Continua)

Apertura, b	13,80 m
Corda di radice, c_r	1,78 m
Rapporto di rastremazione, $\lambda = c_t/c_r$	0,55
Gradiente del coefficiente di portanza del profilo alare, $C_{\ell_{\alpha,w}}$ (dato 2D)	0,105 deg ⁻¹
Svergolamento geometrico d'estremità, ε_t	-1,50 deg
Angolo di portanza nulla dell'ala, $\alpha_{0L,w}$ (dato 3D)	-1,05 deg
Calettamento della corda di radice rispetto alla retta di riferimento della fusoliera, i_w	2,5 deg
Posizione adimensionale del centro aerodinamico dell'ala rispetto al bordo d'attacco della corda media aerodinamica, $X_{ac,w}/\bar{c}$ (dato 3D)	0,28

Tabella 4 Dati caratteristici dell'impennaggio orizzontale. Forma in pianta rettangolare, profilo simmetrico.

Superficie di riferimento, S_H	4,60 m ²
Apertura, b_H	4,60 m
Distanza del centro aerodinamico dell'impennaggio dal centro aerodinamico dell'ala, $X_{ac,H} - X_{ac,w}$	4,90 m
Corda di radice, $c_{r,H}$	1,00 m
Gradiente del coefficiente di portanza del profilo alare, $C_{\ell_{\alpha,H}}$ (dato 2D)	0,11 deg ⁻¹
Fattore di Oswald, e_H (di resistenza indotta)	0,90
Rapporto delle pressioni dinamiche, $\eta_H = \bar{q}_H/\bar{q}_\infty$	0,95
Calettamento dell'impennaggio orizzontale, i_H	-2,0 deg
Fattore di efficacia dell'elevatore, τ_e	0,38
Gradiente del coefficiente di momento di cerniera, $C_{M_{\alpha,e}}$	-0,0076 deg ⁻¹
Gradiente del coefficiente di momento di cerniera, $C_{M_{\delta_e,e}}$	-0,0140 deg ⁻¹
Corda di riferimento dell'equilibratore, $\bar{c}_e$	0,30 m

Tabella 3 (Continua dalla precedente) Dati caratteristici dell'ala.

Coefficiente di momento di beccheggio intorno al centro aerodinamico alare, $C_{M_{ac,w}}$ (dato 3D)	-0,071
Fattore di Oswald, e_w (di resistenza indotta)	0,87
Posizioni adimensionali in apertura delle sezioni estreme degli alettoni, (η_{inner} ; η_{outer})	0,70; 0,95
Fattore di efficacia dell'alettone, τ_a	0,40
Angolo di freccia del bordo d'attacco, Λ_{lc}	12,0 deg
Angolo di diedro, I'	5,0 deg

Tabella 5 Dati caratteristici dell'impennaggio verticale.

Superficie di riferimento, S_V	3,70 m ²
Distanza del centro aerodinamico dell'impennaggio dal baricentro del velivolo, l_V	5,75 m
Distanza verticale media tra il centro aerodinamico dell'impennaggio verticale e la direzione della velocità, h_V	1,35 m
Gradiente del coefficiente di portanza dell'impennaggio, $C_{L_{\alpha,v}}$ (dato 3D)	3,04 rad ⁻¹
Rapporto delle pressioni dinamiche, $\eta_V = \bar{q}_V/\bar{q}_\infty$	1,00
Fattore di efficacia del timone, τ_r	0,48
Gradiente dell'angolo di <i>sidewash</i> , $d\sigma/d\beta$	0,12

Tabella 6 Dati del sistema propulsivo. Si veda la figura 1. (velivolo *bimotore* a elica, propulsori alari).

Diametro dell'elica, D_p	2,2 m
Gradiente dell'angolo di <i>upwash</i> in corrispondenza delle eliche, $d\varepsilon_u/d\alpha_B$	0,420
Gradiente del coefficiente di forza normale dell'elica, $dC_{N_p}/d\alpha_p$	0,0032 deg ⁻¹
Distanza longitudinale del punto di applicazione della spinta dal baricentro, X_T	1,30 m
Distanza laterale del motore destro, Y_T	3,20 m
Distanza verticale del punto di applicazione della spinta dal baricentro, Z_T	0,030 m

FIGURA 4: Esempio di tema d'esame (composto con font Times e MathTimes Professional 2). Tutti i valori numerici in esso presenti sono il risultato dell'espansione di macro definite dall'utente. Le macro vengono assegnate leggendo i dati da un file di testo. Il file di testo viene generato da un foglio di calcolo Mathcad con il quale viene preparata la prova d'esame con il relativo svolgimento. I dati e i risultati del problema svolto sono esportati nel file stampando i valori numerici accanto al nome della macro corrispondente (vengono assegnate circa 300 macro differenti). Il file viene letto successivamente in fase di composizione del documento.

SVOLGIMENTO

L'equazione di equilibrio alla traslazione lungo la normale alla traiettoria va impostata uguagliando la portanza totale $L = L_{WB} + L_H$ al peso. Pertanto si scriverà:

$$L_{WB} + L_H = W \quad (1)$$

con $L_{WB} \approx L_W$.

L'equazione di equilibrio alla rotazione intorno all'asse di beccheggio va impostata uguagliando a zero il momento di beccheggio totale:

$$\mathcal{M}_{WB} + \mathcal{M}_H + \mathcal{M}_{mot} = 0 \quad (2)$$

dove $\mathcal{M}_{mot}$ è il contributo risultante dei propulsori. Le equazioni (1)-(2), opportunamente adimensionalizzate costituiranno un sistema di due equazioni algebriche nelle due incognite α_B e δ_e . Si perviene a tale sistema esplicitando i termini che compaiono nei primi membri, tenendo conto dei dati del problema.

Per la condizione di volo assegnata è possibile calcolare subito il coefficiente di portanza di equilibrio:

$$C_L = \frac{W}{\frac{1}{2}\rho V^2 S} \quad (3)$$

con $\rho = 1,225 \text{ kg/m}^3$, la densità dell'aria al livello del mare. La superficie di riferimento $S \equiv S_W$ dell'ala, data la forma in pianta trapezia, si ottiene come segue:

$$S = \frac{b}{2}(c_r + c_t) = \frac{b}{2}(c_r + \lambda c_r) = \frac{13,80 \text{ m}}{2} 1,78 \text{ m} (1 + 0,55) = 19,0 \text{ m}^2 \quad (4)$$

Pertanto si ha:

$$C_L = \frac{2}{\rho} \frac{W}{S} \frac{1}{V^2} = \frac{2}{1,225 \text{ kg/m}^3} \frac{2500 \text{ kg} \cdot 9,81 \text{ m/s}^2}{19,0 \text{ m}^2} \frac{1}{(55,56 \text{ m/s})^2} = 0,681 \quad (5)$$

Il valore calcolato permetterà più avanti di determinare anche il C_D e la spinta T all'equilibrio.

Conviene calcolare qui la pressione dinamica di volo:

$$q_\infty = \frac{1}{2}\rho V^2 = \frac{1}{2} \cdot 1,225 \text{ kg/m}^3 \cdot (55,56 \text{ m/s})^2 = 1890 \text{ N/m}^2 \quad (6)$$

Calcoli relativi all'ala

Andiamo a determinare le grandezze relative all'ala. Il termine $L_W \approx L_{WB}$ nella (1), quando questo viene adimensionalizzato per $\frac{1}{2}\rho V^2 S$, diventa il coefficiente di portanza dell'ala, dato dall'espressione:

$$(C_L)_{WB} \approx (C_L)_W = (C_{L0})_W + (C_{L\alpha})_W \alpha_B \quad (7)$$

Nella (7) l'angolo α_B è riferito alla retta di riferimento della fusoliera (asse *body* x_B) mentre $(C_{L0})_W$, dato dalla formula:

$$(C_{L0})_W = (C_{L\alpha})_W (i_W - \alpha_{0L,W}) \quad (8)$$

rappresenta il coefficiente di portanza del velivolo parziale in volo a fusoliera orizzontale, cioè quando $\alpha_B = 0$ e l'angolo d'attacco assoluto dell'ala $\alpha_{a,W}$ è pari a $i_W - \alpha_{0L,W}$. Si osservi qui che l'incognita del problema è α_B mentre la velocità di volo V , $\alpha_{0L,W}$ e i_W sono assegnati mentre $(C_{L\alpha})_W$ e $(C_{L0})_W$ sono ricavabili dai dati.

La corda media aerodinamica dell'ala è pari a:

$$\bar{c} = \frac{2}{3}c_r \frac{1 + \lambda + \lambda^2}{1 + \lambda} = 1,42 \text{ m} \quad (9)$$

Un altro parametro necessario ai calcoli è l'allungamento alare $\mathcal{R}_W$, che è pari a:

$$\mathcal{R}_W = \frac{b^2}{S} = 10,0 \quad (10)$$

Ora si può calcolare il gradiente del coefficiente di portanza. Esso si ottiene come segue:

$$(C_{L\alpha})_W = \frac{(C_{L\alpha})_W}{1 + \frac{(C_{L\alpha})_W}{\pi \mathcal{R}_W e_W}} = 4,931 \text{ rad}^{-1} = 0,0861 \text{ deg}^{-1} \quad (11)$$

Per i calcoli precedenti, le grandezze a secondo membro della (8) sono ora tutte note. Dalla (8) si può conoscere quindi il $(C_{L0})_W$ che è pari a:

$$(C_{L0})_W = (C_{L\alpha})_W (i_W - \alpha_{0L,W}) = 4,931 \text{ rad}^{-1} [0,044 \text{ rad} - (-0,018 \text{ rad})] = 0,31 \quad (12)$$

FIGURA 5: Svolgimento del tema d'esame riportato nella figura 5. Anche qui i valori numerici sono il risultato dell'espansione di macro definite dall'utente.

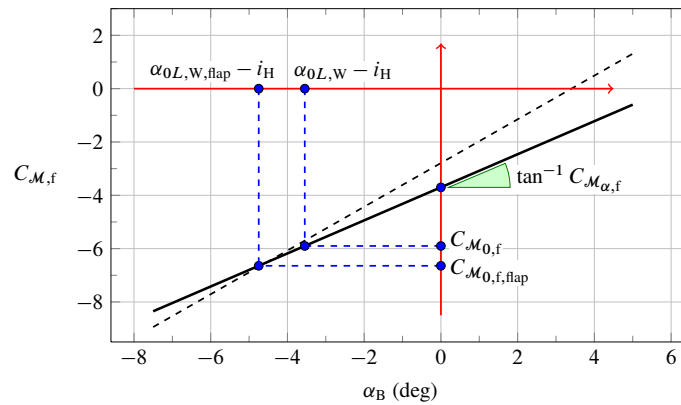


Figura 3 Grafico del momento di beccheggio della fusoliera in presenza dell'ala (curva continua, per la quale $C_{M_{\alpha,f,flap}} \approx C_{M_{\alpha,f}}$). La curva tratteggiata rappresenta un andamento plausibile del $C_{M,f}$ in presenza di un'ala con flap deflessi ($C_{M_{\alpha,f,flap}} > C_{M_{\alpha,f}}$).

caso considerato di piano di coda con forma in pianta rettangolare è

$$S_c = S_H \frac{\bar{c}_c}{c_H} = 4,60 \text{ m}^2 \cdot \frac{0,30 \text{ m}}{1,00 \text{ m}} = 1,4 \text{ m}^2 \quad (50)$$

Pertanto si avrà:

$$\mathcal{H}_{e,A} = 0,0408 \cdot 1890 \text{ N/m}^2 \cdot 1,4 \text{ m}^2 \cdot 0,30 \text{ m} = \underline{31,9 \text{ N m}} \quad (51)$$

FIGURA 6: Estratto dello svolgimento del tema d'esame della figura 5. I valori numerici contenuti nelle macro personalizzate sono stati usati anche per costruire un grafico con il pacchetto `pgfplots`.

```
\expandafter\parseline\@tempa
\fi
\repeat
\closein\myMathcadOutput
}
\makeatother
```

Il comando `\newread` definisce un nuovo canale di lettura chiamato `\myMathcadInput`. Viene poi definita la macro `\parseline` che legge la generica riga catturando la parte #1 che precede il carattere di tabulazione e la parte #2 racchiusa dalle due occorrenze del carattere ". #2 diventa il nome di una nuova macro a cui si assegna valore #1 attraverso il comando `\namedef{#2}{#1}`. Se la macro di nome #2 esiste verrà ridefinita. Infine, si definisce la macro `\parseMathcadOutput` che accetta come argomento il nome del file da leggere applicando `\parseline` ad ogni riga non vuota.

Pertanto, il comando che sostituisce la chiamata a `\input` all'inizio del file `example_1.tex` nella figura 2 è il seguente:

```
\parseMathcadOutput %
{chapter_1/examples/example_1_data.txt}
```

Nella figura 4 è riportato un tema d'esame di Meccanica del volo per gli allievi della laurea in ingegneria aerospaziale dell'Università degli Studi di Napoli "Federico II". Il problema e il relativo svolgimento sono stati impostati con un foglio di calcolo Mathcad Prime 2 di una certa complessità. Il foglio di calcolo crea un file di testo simile a

`example_1_data.txt` ma definisce circa 300 nuove macro.

È questo un esempio di uso avanzato della tecnica di gestione dei numeri proposta in questo articolo. Un tema d'esame come quello mostrato presenta un insieme di dati numeroso. La prova scritta è preparata con cura attraverso il progetto Mathcad verificando rigorosamente l'insieme dei dati del problema, i calcoli intermedi e i risultati finali richiesti. La composizione automatica dei valori numerici assegnati elimina la possibilità di errori manuali che metterebbero in difficoltà gli esaminandi.

La figura 5 riporta la prima pagina dello svolgimento del problema di calcolo, reso pubblico al termine della seduta d'esami. Nella figura 6 si vede un'altra parte dello svolgimento del compito d'esame in cui, sulla base dei valori numerici gestiti con la tecnica descritta, si crea un grafico con il pacchetto `pgfplots`.

La composizione tramite espansione di macro dei valori numerici aiuta in maniera significativa gli esaminatori a preparare lo svolgimento dettagliato del compito d'esame. Gli studenti che si eserciteranno per la prova scritta avranno a disposizione un documento di elevata qualità tipografica, in cui i valori numerici sono affidabili. Nelle prove d'esame successive sarà possibile utilizzare il sorgente `LATEX` della seduta d'esame precedente adattando il testo alle variazioni introdotte. Oppure, nel caso di variazione significativa del tipo di problema

proposto, si potrà usare il progetto Mathcad e il sorgente LATEX come template. Ciò permette agli autori/esaminatori di collezionare problemi svolti che potranno essere forniti agli studenti e adattati per diventare esempi di calcolo in un libro di testo.

4 Generare numeri all'interno del sistema TEX

La gestione dei calcoli può essere effettuata all'interno del processo di composizione con strumenti disponibili in una normale distribuzione del sistema TEX.

Vanno segnalate almeno due possibili vie: usare LuaLATEX (PÈGOURIÈ-GONNARD, 2010) oppure usare il pacchetto di estensione l3fp caricato automaticamente dal pacchetto expl3 (LATEX3 PROJECT, 2012).

4.1 Uso di LuaLATEX

La prima possibilità di gestione dei calcoli tutta interna al mondo TEX è offerta da LuaLATEX. La compilazione con il programma lualatex permette di eseguire blocchi di codice Lua durante la composizione e passare i risultati di programmi di calcolo da Lua a TEX. Come è noto, con l'avvento di LuaLATEX si è 'inoculato' un linguaggio di programmazione moderno e leggero (IERUSALIMSKY, 2006) nel modo LATEX. Potenzialmente LuaLATEX è in grado di assolvere compiti molto complessi ed è certamente un buon candidato alla gestione dei problemi di composizione sollevati dal presente articolo.

Il listato successivo mostra come si gestirebbero in LuaLATEX i valori numerici dell'esempio 1 proposto all'inizio:

```
\documentclass{standalone}
\usepackage{mathtools,array,siunitx}
\usepackage{luacode}% easier catcode management
\begin{luacode*}
function myExample1()
  Vcr = 900*1000/3600; -- m/s
  h = 11000; -- m
  m = 350000; -- kg
  S = 541.2; -- m^2
  rho = 0.364; -- kg/m^3
  W = 9.81; -- m/s^2
  g = m*g; -- N
  Fr = 0.5*rho*S*Vcr^2; -- N
  CL = W/Fr;
  tex.print(
    "\\def\\myCruiseSpeedMTS{.. Vcr .. }"
  )
  tex.print(
    "\\def\\myAltitudeMT{.. h .. }"
  )
  tex.print(
    "\\def\\myMassKG{.. m .. }"
  )
  tex.print(
    "\\def\\myWingSurfaceMTsquared{.. S .. }"
  )
  tex.print(
    "\\def\\myAirDensityKGMTcubed{.. rho .. }"
  )
  tex.print(
```

```
"\\def\\myGravityAccelerationMTSsquared{
  .. g .. }"
)
tex.print(
  "\\def\\myLiftN{.. W .. }"
)
tex.print(
  "\\def\\myReferenceForceN{.. Fr .. }"
)
tex.print(
  "\\def\\myCL{.. CL .. }"
)
end
\end{luacode*}
\begin{document}
\directlua{myExample1()}% definisce le macro
\renewcommand*{\arraystretch}{1.1}
\begin{tabular}{@{}r@{${}=({}$}1@{}}
\hline
$m$
& \SI[round-precision=0]{\myMassKG}{kg}
\\
$$
& \SI[round-precision=1]
  {\myWingSurfaceMTsquared}{m^2}
\\
$V_{\mathrm{cr}}$
& \SI[round-precision=3]{\myCruiseSpeedMTS}{m/s}
\\
$h$
& \SI[round-precision=0]{\myAltitudeMT}{m}
\\
$\rho$
& \SI[round-precision=3]
  {\myAirDensityKGMTcubed}{kg/m^3}
\\
$F_{\mathrm{r}}$
& \SI[round-mode=figures,
  scientific-notation=engineering,
  round-precision=3]{\myReferenceForceN}{N}
\\
$L$
& \SI[round-mode=figures,
  scientific-notation=engineering,
  round-precision=3]{\myLiftN}{N}
\\
$C_L$
& \SI[round-precision=3]{\myCL}{ }
\\
\hline
\end{tabular}
\end{document}
```

Il codice precedente va compilato con il programma lualatex e produce la seguente tabella:

m	$= 350000 \text{ kg}$
S	$= 541,2 \text{ m}^2$
V_{cr}	$= 250 \text{ m/s}$
h	$= 11000 \text{ m}$
ρ	$= 0,364 \text{ kg/m}^3$
F_{r}	$= 6,16 \cdot 10^6 \text{ N}$
L	$= 3,43 \cdot 10^6 \text{ N}$
C_L	$= 0,558$

nella quale, ancora, i valori numerici sono stati parametrizzati con delle macro.

Nel sorgente LATEX è stato inserito del codice Lua all'interno di un ambiente luacode. Il codice Lua definisce la funzione myExample1() che non accetta argomenti. Si noti che l'ambiente luacode può essere inserito anche nel preambolo.

La funzione `myExample1()` definisce le variabili necessarie ed effettua i semplici calcoli richiesti. Infine, tramite diverse chiamate alla funzione `tex.print` passa a $\text{\TeX}$ delle stringhe formattate (corrispondenti al codice del file `example_1_data.tex` discusso in precedenza).

Nel codice che genera la tabella, all’interno dell’ambiente `document`, il controllo passa da $\text{\TeX}$ a Lua attraverso la macro `\directlua`. L’argomento di `\directlua` è semplicemente la funzione `myExample1()`, pensata per generare numeri, definire macro e comunicarle al motore di composizione. Dopo l’istruzione `\directlua{myExample1()}` le macro con i valori numerici restano definite e possono essere usate all’interno dei comandi `\SI`.

4.2 Uso del pacchetto l3fp

Un’alternativa a $\text{\LaTeX}$ è data dal pacchetto `l3fp` di $\text{\LaTeX}$ 3 che permette di gestire espressioni in virgola mobile (floating point) usando `pdflatex`. Il codice seguente ne costituisce un semplice esempio d’uso:

```
\usepackage{expl3}% nel preamble
% ...

\ExplSyntaxOn
\fp_set:Nn \mynumber { sin ( pi/4 ) }
\[
\sin \frac{\pi}{4} =
\quad \num[round-mode=places,
\quad round-precision=3]{\fp_use:N \mynumber }
\]
\ExplSyntaxOff
```

che produce l’espressione in display:

$$\sin \frac{\pi}{4} = 0,707$$

Si osservi che i comandi di `expl3` devono essere racchiusi dalle macro `\ExplSyntaxOn` e `\ExplSyntaxOff`.

Una possibile gestione in $\text{\LaTeX}$ 3 dell’esempio di calcolo di questo articolo è la seguente:

```
\documentclass{article}

\usepackage{expl3}% nel preamble
% ...

\ExplSyntaxOn
% ISA density vs altitude
\cs_new:Npn \fn_rho_ISA #1
{1.225*((288.16 - 0.0065*#1)/288.16)^(4.257)}

\newcommand\myExampleFP{%
\fp_new:N \myMassKG
\fp_set:Nn \myMassKG{350000}
\fp_new:N \myAltitudeMT
\fp_set:Nn \myAltitudeMT{11000}
\fp_new:N \myCruiseSpeedMTS
\fp_set:Nn \myCruiseSpeedMTS{900 *1000/3600}
\fp_new:N \myWingSurfaceMTsquared
\fp_set:Nn \myWingSurfaceMTsquared{541.2}
\fp_new:N \myLiftN
\fp_set:Nn \myLiftN{ \fp_use:N \myMassKG * 9.81 }
\fp_new:N \myAirDensityKGMTcubed
\fp_set:Nn \myAirDensityKGMTcubed{ \fn_rho_ISA{
\fp_use:N \myAltitudeMT } }
```

```
\fp_new:N \myReferenceForceN
\fp_set:Nn \myReferenceForceN{
0.5 * \fp_use:N \myAirDensityKGMTcubed
* \fp_use:N \myCruiseSpeedMTS ^2
* \fp_use:N \myWingSurfaceMTsquared }
\fp_new:N \myCL
\fp_set:Nn \myCL{
\fp_use:N \myLiftN
/ \fp_use:N \myReferenceForceN }
}
% simili alle macro di siunitx, ma capaci
% di gestire espressioni
\NewDocumentCommand{ \calcnun } { o m }
{ \num[#1]{ \fp_to_scientific:n {#2} } }
\NewDocumentCommand{ \calcSI } { o m m }
{ \SI[#1]{ \fp_to_scientific:n {#2} }{#3} }
\ExplSyntaxOff

\begin{document}
\myExampleFP% definisce le macro contenitore

\renewcommand*{\arraystretch}{1.1}
\begin{tabular}{@{}r@{${}={}$}l@{}}
\hline
$m$
& \calcSI[round-precision=0]{\myMassKG}{kg}
\\
$$
& \calcSI[round-precision=1]
{ \myWingSurfaceMTsquared }{m^2}
\\
$V_\mathrm{cr}$
& \calcSI[round-precision=3]
{ \myCruiseSpeedMTS }{m/s}
\\
$h$
& \calcSI[round-precision=0]{\myAltitudeMT}{m}
\\
$\rho$
& \calcSI[round-precision=3]
{ \myAirDensityKGMTcubed }{kg/m^3}
\\
$F_\mathrm{r}$
& \calcSI[round-mode=figures,
scientific-notation=engineering,
round-precision=3]
{ \myReferenceForceN }{N}
\\
$L$
& \calcSI[round-mode=figures,
scientific-notation=engineering,
round-precision=3]
{ \myLiftN }{N}
\\
$C_L$
& \calcSI[round-precision=3]{\myCL}{ }
\hline
\end{tabular}
\end{document}
```

Questo sorgente compone la stessa tabella riprodotta con $\text{\LaTeX}$ al paragrafo precedente. Si esamini la macro `\myExampleFP` che definisce e assegna le variabili contenenti i dati attraverso i comandi `fp\_new:N` e `fp\_set:Nn` di `l3fp`. Questa macro valuta anche una funzione `fn_rho_ISA` definita dall’utente per assegnare il valore della densità dell’aria alla quota assegnata.

Nel preamble del documento sono definite due macro di utilità `\calcnun` e `\calcSI` con la sintassi di $\text{\LaTeX}$ 3. Esse si basano sulle macro `\num` e `\SI` di `siunitx` ed accettano espressioni numeriche.

L'esempio precedente è solo un accenno alle recenti funzionalità di calcolo in virgola mobile introdotte dagli sviluppatori di L^AT_EX3. Approfondire i possibili miglioramenti a queste tecniche di codifica del codice basato su `expl3` esula dagli scopi del presente articolo. Ciò sarà probabilmente oggetto di un futuro articolo.

5 Conclusioni

Si è presentata una strategia di organizzazione del codice L^AT_EX mirata alla preparazione di esempi di calcolo in un testo tecnico-scientifico, con gestione dei dati effettuata attraverso software di calcolo esterni.

Si sono presentati due casi in cui lo strumento di lavoro esterno scelto dall'autore è più (Matlab) o meno (Mathcad Prime) flessibile in termini di esportazione dei dati in file di testo.

Queste tecniche sono anche alla portata di utenti L^AT_EX meno esperti e si prestano agevolmente alla gestione di un numero elevato di esempi ed esercizi svolti in manoscritti di grandi dimensioni corredati da software fornito dall'autore.

Infine, si sono riportati esempi di codice LuaL^AT_EX e L^AT_EX3 in grado di gestire automaticamente semplici problemi di calcolo.

Riferimenti bibliografici

L^AT_EX3 PROJECT (2012). «The L^AT_EX3 interfaces». Disponibile all'indirizzo

<http://texdoc.net/texmf-dist/doc/latex/l3kernel/interface3.pdf>.

BECCARI, C. (2012). «Introduzione all'arte della composizione tipografica con L^AT_EX». G_UI_TE_X, Gruppo Utilizzatori Italiani di T_EX. Disponibile all'indirizzo: <http://www.guitex.org/home/images/doc/guidaguit.pdf>.

IERUSALIMSKY, R. (2006). *Programming in Lua*. Lua.org, 2^a edizione.

PÈGOURIÈ-GONNARD, M. (2010). «A guide to LuaL^AT_EX». Disponibile all'indirizzo <http://texdoc.net/texmf-dist/doc/lualatex/lualatex-doc/lualatex-doc.pdf>.

— (2012). «The luacode package version 1.2a». Disponibile su www.ctan.org.

WESSELINGH, H. e DE WAARD, H. (2012). *Calculate & Communicate with Mathcad Prime*. VSSD, 1^a edizione.

▷ Agostino De Marco
Università degli Studi di Napoli
"Federico II"
Dipartimento di Ingegneria Aerospaziale
agostino.dot.demarco@unina.it

Il pacchetto esami per la creazione di prove scritte*

Grazia Messineo, Salvatore Vassallo

Sommario

Viene presentato il pacchetto **esami** che implementa alcune caratteristiche dei pacchetti **exerquiz** e **probsoln** per la produzione collaborativa di database di esercizi e la produzione di temi d'esame.

Abstract

We present the package **esami** which extends some useful properties of the $\text{\LaTeX}$ packages **exerquiz** and **probsoln** to produce databases of exercises in a collaborative way and to produce exams.

1 Introduzione

In questo lavoro viene presentato un pacchetto da noi elaborato per la gestione delle prove d'esame di Matematica Generale dei corsi della Facoltà di Economia dell'Università Cattolica.

Il pacchetto, pur con alcune modifiche, è in uso da alcuni anni e si è dimostrato stabile ed affidabile, ha permesso di generare senza problemi 60 varianti diverse di uno stesso compito, è in grado di gestire grafici dipendenti da parametri e si è dimostrato flessibile così da essere impiegato in situazioni molto diverse, dalla creazione di prove di verifica nelle scuole secondarie (assegnando ad ogni studente una prova diversa), alla gestione di esami universitari con 20 esercizi.

All'inizio degli anni '90 in Università Cattolica ci si è trovati a dover affrontare il problema di far sostenere prove scritte di matematica ad un numero molto elevato di studenti in aule di capienza insufficiente per impedire "comunicazioni".

Ci si è quindi posti l'obiettivo di creare, con l'ausilio del computer, un numero teoricamente illimitato di varianti di un compito base, varianti in cui gli esercizi dovevano essere "simili", ma con valori numerici diversi, un ordinamento diverso e, nel caso di domande a risposta chiusa, con l'ordinamento delle risposte diverso.

Nel corso degli anni sono stati fatti numerosi tentativi per raggiungere questo obiettivo, usando in tutti i casi programmi a pagamento:

- la creazione di un insieme di macro scritte in *Visual Basic* di Microsoft Word, che estraevano casualmente gli esercizi da un insieme, assegnavano sempre in modo aleatorio dei valori ai

parametri presenti negli esercizi, riordinavano domande e risposte. Il metodo usato presentava alcuni problemi e, alla fine, un cambiamento nel linguaggio di programmazione nella suite Office ha reso inutilizzabili tali macro;

- l'uso di Scientific WorkPlace, in particolare del suo generatore di esami con caratteristiche molto simili a quelle desiderate e del modulo che permette di utilizzare un sottoinsieme degli strumenti matematici di Maple o Mupad. I problemi di questa soluzione erano di tre tipi: la formattazione non ottimale dovuta al fatto che il modulo per la creazione degli esami non utilizza $\text{\LaTeX}$, ma è più simile al linguaggio RTF, una certa limitatezza nei grafici creati (ad esempio non era possibile inserire parametri) e, soprattutto, il fatto che l'algoritmo per la generazione delle varianti non era noto e quindi poteva accadere facilmente che due compiti "vicini" fossero molto simili tra loro.

2 La filosofia del pacchetto

2.1 La scelta di $\text{\LaTeX}$

I problemi sopra esposti hanno spinto a cercare un'altra soluzione e la si è individuata nell'utilizzo completo di $\text{\LaTeX}$.

I vantaggi della scelta di $\text{\LaTeX}$ sono molteplici:

1. la modularità;
2. la possibilità di modificare il codice, controllare a priori l'output delle routine usate, implementare delle proprie macro;
3. le sue capacità di calcolo e grafiche;
4. la possibilità di modificare molti aspetti di formattazione senza modificare il testo;
5. l'output in PDF e quindi la portabilità;
6. la gratuità del software e quindi il risparmio sui costi delle licenze.

L'unico aspetto negativo è che la curva di apprendimento di $\text{\LaTeX}$ ha una notevole pendenza iniziale e quindi è richiesto agli utilizzatori un periodo di apprendimento abbastanza impegnativo.

Una rapida ricerca sul CTAN mostra che esistono già molti pacchetti e classi dedicati alla stesura di esami, alcuni dei quali sono già stati illustrati su $\text{\LaTeX}$ nica (PENNA, 2006; PILU, 2011).

*Il lavoro si inserisce nel progetto di ricerca di interesse d'Ateneo dell'Università Cattolica "Progetto M.In.E.R.Va: Creazione di esercizi di tipo interattivo con percorsi differenziati per il recupero delle carenze e la valorizzazione delle eccellenze in matematica".

2.2 Il pacchetto *exerquiz*

La scelta iniziale è stata quella di utilizzare il pacchetto *exerquiz* (STORY, 2012) che permette di creare degli esami con domande a risposta chiusa (con la possibilità di variare casualmente l'ordine delle risposte) o aperta; tali esercizi possono essere suddivisi in più parti. All'interno del file con gli esercizi sono presenti anche le soluzioni e tramite un'opzione si può decidere di visualizzarle o meno.

Una delle caratteristiche più importanti del pacchetto è di permettere facilmente l'incorporazione di codice JavaScript nel file PDF di output. Tale funzionalità permette un certa interattività del file PDF come, ad esempio, la correzione automatica degli esercizi: tale caratteristica è stata ampiamente sfruttata in altre occasioni (MESSINEO e VASSALLO, 2012).

Nelle nostre intenzioni un compito era una creatura abbastanza complessa, composta da esercizi a risposta chiusa e a risposta aperta (problemi). Supponendo che i docenti scelgano 10 esercizi (come nel nostro caso), ogni esercizio (sia esso a risposta chiusa o aperta) ha un numero proprio di varianti e ogni variante può dipendere da alcuni parametri. La generazione di un compito prevede che, per ogni versione prodotta (generalmente da 20 a 40 versioni) il pacchetto scelga da ognuno dei file una variante, mescoli i 10 esercizi scelti, nel caso di domande a risposta chiusa mescoli le risposte, scelga i valori dei parametri di ogni esercizio e infine generi un file con le risposte corrette per ogni versione.

Era necessario perciò che il pacchetto *exerquiz* venisse integrato, poiché tra le caratteristiche volute ha solo la possibilità di permutare casualmente le risposte nelle domande a risposta chiusa. Sia per questo motivo, sia per avere del software “stabile” nel tempo, si è deciso di costruire, partendo da un sottoinsieme di *exerquiz*, un pacchetto autonomo che abbiamo chiamato *esami*. Nel corso degli anni (la prima versione del pacchetto è del 2007) ci sono state poi molte intersezioni tra il nostro lavoro e quello dell'autore di *exerquiz*, con soluzioni a volte simili, a volte diverse.

Il problema di utilizzare parametri all'interno degli esercizi è di facile soluzione: infatti esistono pacchetti che implementano la generazione di numeri casuali e la cosa è possibile anche con un comando nativo di pdf_LA_TE_X (il comando `\pdfuniformdeviate`). La nostra scelta è stata di utilizzare il pacchetto *random* (ARSENEAU). Si è deciso poi di utilizzare il pacchetto *fp* (MEHLICH, 1999) per le operazioni matematiche che devono essere effettuate sui parametri perché ha una notazione più intuitiva e ci ha reso possibile implementare altri comandi per semplificare la scrittura degli esercizi.

Il problema della scelta della variante di un esercizio tra quelle proposte è stato risolto adattando e semplificando un comando del pacchetto *prob-*

soln (TALBOT, 2011): esso è stato creato per la generazione di raccolte di esercizi, scelti tra diverse varianti, da presentare agli studenti in anni accademici diversi.

Abbiamo creato infine un file di configurazione modificabile dall'utente che contiene materiale accessorio (come l'intestazione del compito, le istruzioni per lo studente, ecc.) e dei *template* in cui devono essere inserite variabili che competono direttamente alla generazione del compito, quali il numero di versioni da generare, il numero di esercizi nelle varie parti del compito, la data, ecc..

2.3 Le modifiche al codice

Già dalle prime generazioni di esercizi, il pacchetto *esami* ha subito evidenziato alcuni limiti.

Il primo limite era intrinseco alla costruzione fatta: era infatti ancora possibile che versioni “vicine” fossero troppo simili. Tale problema nasceva dal fatto che, nel nostro caso, una parte del compito era composta da tre soli esercizi: poiché esistono 6 permutazioni di 3 elementi, la probabilità che versioni adiacenti avessero esercizi simili e nello stesso ordine era di 1/6. Il secondo problema era invece legato alla parametrizzazione degli esercizi: per ogni parametro presente in un esercizio veniva assegnato un intervallo di variazione in cui veniva scelto aleatoriamente il valore. Il seme per la scelta aleatoria viene però fissato all'inizio dell'elaborazione del compito (attualmente dipende dalla data e dal numero della versione del compito); in tale modo parametri con lo stesso intervallo di variazione erano sempre uguali. Il terzo problema era la creazione di una stringa di soluzioni per la correzione del compito e, più in generale, della soluzione degli esercizi: se infatti il pacchetto *exerquiz* permette facilmente la creazione di una versione del compito con le soluzioni indicate, era di sicuro preferibile avere a disposizione la stringa delle soluzioni alle domande a risposta chiusa. La creazione di tale stringa, visto anche il procedimento di generazione del compito, non si presentava di facile soluzione; inoltre *exerquiz* scrive le soluzioni delle domande a risposta aperta *verbatim* su un file esterno e le include o no a seconda dell'opzione scelta; tale possibilità ci era preclusa poiché i nostri esercizi sono argomento di un comando e quindi non è possibile usare codice *verbatim*¹.

Un ulteriore problema era invece legato ai file contenenti i sorgenti degli esercizi e, in particolare, alla fase preliminare di controllo della correttezza di tali esercizi: è infatti opportuno avere a disposizione le versioni dei file scritte in forma parametrica, ma non in codice T_EX in modo da permettere a chiunque il controllo dei testi.

1. Il pacchetto *cprotect* risolve tale problema e abbiamo utilizzato tale soluzione in un altro ambito; tale pacchetto è però più recente del nostro e quando l'abbiamo scoperto abbiamo deciso di mantenere il codice che aveva già dimostrato di funzionare bene.

Le nostre soluzioni sono state:

- (a) La scelta delle varianti per versioni vicine: come detto con una scelta puramente casuale la probabilità di compiti vicini simili può essere troppo alta (anche se la struttura del compito permette altri “mascheramenti”). Per ovviare a tale inconveniente la nostra soluzione è stata quella di avere sia per la scelta della variante all’interno di un esercizio sia per le permutazioni degli esercizi all’interno del compito, un’assegnazione deterministica da un sottoinsieme delle permutazioni possibili se il numero di “oggetti” tra cui scegliere è piccolo e una scelta del tutto aleatoria altrimenti.
- (b) Al problema della variazione dei parametri si è ovviato da un lato valutando con maggiore attenzione nella stesura degli esercizi l’intervallo di variazione dei parametri, dall’altro inserendo nel codice una routine per far scegliere un numero successivo nella sequenza dei numeri casuali; vi è inoltre la possibilità di una variazione “locale” del seme, per cui se il parametro a è nell’intervallo $[\alpha, \beta]$ si può richiedere di generare il parametro b nello stesso intervallo, ma con un seme diverso (e quindi ottenendo, in generale, un valore diverso). Infine c’è la possibilità di generare numeri casuali in un intervallo che abbia come estremi parametri (già assegnati) o che non assuma valori all’interno un insieme assegnato².
- (c) La soluzione al problema della creazione della stringa delle soluzioni è stata quella di sfruttare il sistema di riferimenti incrociati `\label` e `\ref` e il file `aux`, mentre per le soluzioni delle domande a risposta aperta si è creato un ambiente `solution` che scrive normalmente sul file di output se si ha l’opzione `solutions` mentre cancella il suo contenuto con l’opzione `nosolutions`.

Della soluzione all’altro problema prospettato (visualizzazione degli esercizi in forma parametrica) ci si occuperà più avanti, presentando i file che il pacchetto usa.

Ci sono poi altri problemi di natura diversa: sono possibili di sicuro notevoli miglioramenti nel codice e nella gestione delle liste; la soluzione da noi implementata per avere un file esterno di soluzioni di facile lettura impedisce di usare domande a risposta chiusa multipla e, infine, poiché si è deciso di inserire ogni esercizio in `minipage`³ non sempre la formattazione è ottimale.

2. Il programma cercherà di soddisfare tutte le richieste generando nuovamente il parametro per un certo numero di volte, limitato da una quantità fissata dall’utente.

3. Alcuni studenti si erano lamentati di non avere visto mezzo esercizio perché diviso su due pagine, per esempio.

I problemi maggiori sono però “umani” e riguardano la maggiore attenzione necessaria nell’elaborazione e nel controllo degli esercizi, soprattutto a risposta chiusa. Non è infrequente infatti che per alcuni valori dei parametri ci possano essere due risposte uguali o incoerenti. Questa maggiore attenzione viene però scontata in una maggiore facilità e rapidità di correzione delle prove.

2.4 Le tipologie di esercizi

Per i nostri scopi iniziali erano necessari solo due tipi di esercizi: i “test”, cioè insiemi di esercizi a risposta chiusa, e i “problemi”, cioè esercizi a risposta aperta con uno svolgimento articolato, come uno studio di funzione o la discussione e soluzione di un sistema lineare.

In vista della possibilità di usare lo stesso software anche in situazioni diverse si è però deciso fin da subito di non limitare la possibilità di scelta degli esercizi a quelle sopra citate, ma di implementarne altri tipi⁴:

- (a) domande a risposta aperta “breve” in cui il risultato viene scritto in un apposito spazio alla fine dell’esercizio;
- (b) “riempi lo spazio (*fillin*)”: esercizi in cui devono essere riempiti alcuni spazi lasciati bianchi in proposizioni, teoremi, ecc.
- (c) domande a risposta aperta “lunga”;
- (d) tabelle da completare;
- (e) “*matching*”: esercizi in cui bisogna collegare gli elementi di due liste come, ad esempio, una lista di funzioni con quella delle loro derivate.

Ovviamente tutti questi esercizi possono contenere parametri aleatori, possono essere riordinati casualmente e infine possono avere diverse apparenze “estetiche” (ad esempio lo spazio per la risposta può essere un riquadro, una riga sottolineata, ecc.)

Riuscire ad avere tutte queste caratteristiche, oltre a quelle relative al foglio soluzioni, ha comportato diverse difficoltà e a questo proposito vogliamo ringraziare il `qJr` e in particolare il prof. Enrico Gregorio per i preziosi suggerimenti e consigli.

2.5 Il problema della randomizzazione: le nostre scelte

In `LaTeX` esistono diversi pacchetti che generano numeri casuali (per l’esattezza, pseudocasuali).

Come già detto la nostra scelta è stata quella di affidarci a un metodo “misto”. La permutazione delle risposte chiuse ad una domanda è del tutto aleatoria così come la scelta dei parametri. Invece la scelta di una variante di un esercizio tra quelle proposte è aleatoria se il numero di varianti è

4. In particolare alcune tipologie sono nate da precise richieste in ambito di scuola secondaria per la gestione di prove di verifica.

maggiore di un valore prestabilito (attualmente è fissato a 8), mentre negli altri casi viene scelta (deterministicamente sulla base del numero di versione del compito) una permutazione tra 24 possibili permutazioni (6 nel caso le varianti siano 3). Un procedimento analogo viene utilizzato anche per la scelta dell'ordine degli esercizi nel compito.

Nel processo di randomizzazione il seme è stato scelto con una combinazione della data dell'esame e del numero della versione così che compiti "identici" assegnati in date diverse siano differenti. Inoltre nella generazione dei parametri aleatori interviene anche il numero d'ordine dell'esercizio nel compito generato, così che parametri definiti nello stesso intervallo, ma in esercizi diversi, siano differenti.

2.6 La struttura di un esame

Per la generazione di una prova d'esame è necessaria la creazione da parte dell'utente di un certo numero di file:

- (a) I file contenenti i testi di ogni esercizio, ognuno con il suo numero di varianti.
- (a) Un file "master" per la prova d'esame in cui l'utente interviene solo per indicare la data della prova, il numero di versioni da generare e, ovviamente, gli esercizi da inserire⁵.
- (b) Un file "master" per le soluzioni, identico al precedente, che genera le soluzioni per ogni versione della prova e la stringa delle risposte alle domande a risposta chiusa.

Vi è inoltre un altro file utilizzato per controllare il testo degli esercizi: per ogni esercizio, stampa tutte le varianti sia in versione parametrica che in una versione numerica. Ciò è stato possibile grazie a un comando che, in dipendenza di un'opzione del pacchetto, calcola o meno le espressioni matematiche presenti nell'esercizio.

Nel corso dell'elaborazione del pacchetto abbiamo anche implementato comandi per la semplificazione di frazioni, di radicali⁶, per la formattazione degli esponenti, ecc..

Compito dei docenti è quindi quello di creare un database di esercizi, controllarli e scriverli in L^AT_EX utilizzando le istruzioni del software. Dopo di che, in previsione di una prova d'esame, vengono scelti gli esercizi e generati i compiti.

Oltre a questi file modificabili dall'utente, sono stati creati altri file che non devono in generale essere modificati dall'utente finale: il file `esami.sty` e un file, modificabile avendo un po' di conoscenza del linguaggio L^AT_EX, contenente le configurazioni "estetiche" quali il formato della testata, del piè di

5. In questo file è possibile modificare anche alcuni parametri che determinano l'aspetto dell'esame, ma per questo è necessaria una (minima) conoscenza di L^AT_EX.

6. Il risultato non è però ulteriormente semplificabile per cui, ad esempio, $\sqrt{8}/2$ si riduce a $2\sqrt{2}/2$

pagina e le istruzioni per lo studente che appaiono sul foglio dell'esame.

Le motivazioni per questo tipo di struttura sono molteplici:

- (a) la creazione e il controllo degli esercizi è del tutto svincolata dalla generazione di un tema d'esame anche se il formato del file è identico: ciò permette la creazione di un vero e proprio database di esercizi che può essere incrementato quando se ne ha la possibilità;
- (b) le modifiche "stilistiche" al tema d'esame possono essere effettuate senza bisogno di mettere mano al vero e proprio set di istruzioni;
- (c) un utente con poche, se non nulle, conoscenze di L^AT_EX può essere in grado di generare il tema d'esame una volta che il database è pronto;
- (d) poche modifiche al file master permettono di gestire situazioni molto diverse (esami in una o più parti, numero di domande diverso, diverse tipologie di esercizio, ecc.).

3 Il pacchetto

Come detto del pacchetto fanno parte un file di stile, un file di configurazione, due file master, un file per il controllo degli esercizi e, ovviamente, i file che costituiscono il database degli esercizi. Cominciamo da questi ultimi.

3.1 I file degli esercizi

Ogni esercizio (con tutte le sue varianti) va scritto in un file separato. Ogni variante è racchiusa nel comando `\newproblem` che è una versione (molto) modificata dell'analogo comando del pacchetto `probsoln`, tale comando ha come unico argomento il testo dell'esercizio.

Esercizi a risposta chiusa

Se l'esercizio è a risposta chiusa la sintassi è:

```
\item \PTs{punteggio}
... Testo ...
\begin{answers}{numero colonne}
  \bChoices[random]
  \Ans0 risposta errata \eAns
  \Ans0 risposta errata \eAns
  \Ans1 risposta esatta \eAns
  \eFreeze
  \Ans0 nessuna delle precedenti \eAns
  \eChoices
\end{answers}
```

nella quale:

- `\item \PTs{punteggio}` introduce una domanda con punteggio indicato in `\PTs` (può essere anche un numero decimale e il separatore può essere la virgola a differenza del pacchetto `exerquiz`);

- `\begin{answers}{numero colonne}`
`\bChoices[random]`
`...`
`\eChoices`
`\end{answers}`
 introduce le risposte disposte sul numero di colonne specificate. Le risposte vengono permutate in modo casuale solo se è specificata l'opzione `random`;
- `\Ans0` introduce una risposta errata;
- `\Ans1` introduce la risposta esatta⁷;
- `\eFreeze` introduce (se si vuole) una o più risposte che non saranno in ordine casuale.

Esercizi a risposta aperta

Se l'esercizio è un problema a risposta aperta verrà inserito nell'ambiente `problem` o `problem*` se ha una o, rispettivamente, più parti. La sintassi sarà:

```
\begin{problem} [punteggio]
... Testo ...
\begin{solution}[spazio_soluzione]
... Soluzione ...
\end{solution}
\end{problem}
```

In `punteggio` va il punteggio dell'esercizio, in `spazio_soluzione` va l'eventuale spazio bianco che deve essere lasciato per la soluzione. Oppure nel caso di esercizio in più parti:

```
\begin{problem*} [punteggio]
....testo....
\begin{parts}
\item \PTs{punteggio della parte}
....testo....
\begin{solution}[spazio_soluzione]
.... testo soluzione .....
\end{solution}
\item \PTs{punteggio della parte}
.....
\end{parts} \end{problem*}
```

dove con `\PTs{punteggio della parte}` si indica il punteggio della singola parte. Mentre il pacchetto `exerquiz` ha la possibilità di calcolare automaticamente il punteggio dell'esercizio con l'argomento `[<\auto>]`, l'introduzione della randomizzazione della scelta degli esercizi in `esami` ha tolto tale possibilità.

Altre tipologie di esercizi

Sono state definite alcune nuove tipologie di esercizio:

7. Il meccanismo adottato per riportare in un file esterno le soluzioni ha purtroppo escluso, per ora, la possibilità di avere domande con più di una risposta esatta

fillin Serve per creare esercizi "a riempimento": parte del testo viene lasciata vuota e deve essere riempita dallo studente. Può essere usato anche per creare esercizi a risposta aperta breve. La sintassi è `\fillin[<tipo>]{<larghezza dello spazio>}{<risposta>}`. I due parametri obbligatori sono la larghezza dello spazio da lasciare, che deve essere espressa in cm, e la risposta esatta (parola o numero) che lo studente dovrebbe inserire (e che verrà stampata solo nel file delle soluzioni). Il parametro opzionale `<tipo>` definisce come deve essere segnalato lo spazio in cui inserire la risposta: `u` (*underlined*), che è l'opzione di default, fa sì che lo spazio sia evidenziato mediante una riga su cui va inserita la risposta, `e` (*empty*) crea uno spazio vuoto (non evidenziato in alcun modo), `b` (*boxed*) crea uno spazio circondato da un riquadro.

All'interno del comando `\fillin` (nello spazio per la risposta) non è possibile usare i comandi di semplificazione delle frazioni (si veda il paragrafo 3.4).

matching Serve per creare esercizi in cui le parole o formule contenute su due colonne vanno abbinate dallo studente in modo corretto. Le coppie vengono definite dal comando `\pair`: `\pair{<nome 1>}{<nome 2>}` che va inserito tante volte quanti sono i nomi o le formule da abbinare. L'elenco dei nomi (mischiati in ordine casuale su ciascuna colonna) viene visualizzato con il comando `\matching`. Ad esempio:

```
\pair{Italia}{Roma}
\pair{Germania}{Berlino}
\pair{Grecia}{Atene}
\matching
```

Grecia	Berlino
Italia	Atene
Germania	Roma

Nelle soluzioni viene invece visualizzato l'abbinamento corretto.

tabella serve per creare esercizi con più risposte aperte incolonnate. La sintassi è:

```
\begin{tabella}[num. col. visibili]
{allineamento colonne visibili}
{allineamento colonna nascosta}
... & ... \cr
\end{tabella}
```

Il primo parametro (il cui valore di default è 2) indica il numero di colonne della tabella il cui contenuto deve essere visibile anche nel testo dell'esercizio (e non solo nella soluzione), mentre è possibile avere solo una colonna il cui contenuto è invisibile nel testo ma appare nella soluzione. Il secondo parametro consente di scegliere l'allineamento delle

colonne il cui contenuto è sempre visibile e il terzo l'allineamento della colonna invisibile.

Ad esempio, con il codice:

```
\begin{center}
\renewcommand\arraystretch{3}
\begin{tabella}[1]{1}{1}
\hline
Il dominio della funzione è &
$D_f=(-\infty;2]$ \cr
\hline
L'insieme immagine di $f(x)$ è &
$Im(f)=(-\infty,0]$ \cr
\hline
\end{tabella}
\end{center}
```

si ottiene il seguente risultato (la seconda colonna è visibile solo nelle soluzioni):

Il dominio della funzione è	$D_f = (-\infty; 2]$
L'insieme immagine di $f(x)$ è	$Im(f) = (-\infty, 0]$

I successivi ambienti non definiscono una tipologia vera e propria di esercizi.

problema e problema* Servono per creare esercizi a risposta aperta. Il funzionamento è analogo a **problem** e **problem***, la differenza sta nel fatto di poter eliminare (mediante l'opzione **solutionsonly**) il testo degli esercizi, riportando solo la soluzione.

risposta Serve per creare, in esercizi a risposta aperta lunga, lo spazio per inserire la risposta in un box o su righe. La sintassi del comando è:

```
\begin{risposta}{tipo}{spazio verticale}
...
\end{risposta}
```

Il primo parametro (**tipo**) definisce se lo spazio per la risposta deve essere riquadrato (opzione **b**, di default) oppure diviso in righe su cui andrà scritta la risposta (opzione **l**). Il secondo parametro definisce invece l'altezza dello spazio per la risposta (in cm se con opzione **b**, in numero di righe se con opzione **l**).

workarea serve a creare un'area di lavoro, cioè uno spazio bianco sul foglio in cui lo studente possa scrivere. La sintassi è:

```
\begin{solution}{lunghezza}
\end{solution}
\begin{workarea}{larghezza}{lunghezza}
\end{workarea}
```

La lunghezza dello spazio indicato per **solution** e della **workarea** dovrebbero essere uguali; in

caso contrario il testo presente nella **workarea** viene posizionato male nello spazio della soluzione, sovrapponendosi al testo dell'esercizio se la sua lunghezza è maggiore di quella della soluzione. La larghezza della **workarea** è opzionale e di default è uguale alla larghezza del testo. A differenza dell'ambiente **solution**, nella **workarea** è possibile inserire testo, grafici, ecc.; ad esempio, è possibile inserire degli assi coordinati per disegnare un grafico.

3.2 I file master

I file master e master-sol

Questi due file differiscono solo perché il secondo mostra le soluzioni e contengono le istruzioni per generare effettivamente la prova d'esame. In entrambi è necessario indicare la data (comando **\date**) in formato $\langle \text{giorno} \rangle / \langle \text{mese} \rangle / \langle \text{anno} \rangle$ (giorno e mese possono essere scritti in qualsiasi formato: 3/12/2012, 03/7/2013; l'anno deve essere indicato con 4 cifre), il numero di compiti da generare (comando **\numcompiti**) e gli esercizi. Il tema d'esame può essere diviso in più parti e in ognuna si può usare uno o più degli ambienti introdotti e uno o più comandi per la stampa degli esercizi. Nel file possono essere definiti anche il numero massimo di ripetizioni che il comando **\FPsetpar** (si veda 3.4) deve eseguire per cercare di soddisfare le condizioni poste (comando **\maxLoopLimit**) e il meccanismo di calcolo del seme iniziale per la generazione dei numeri casuali (comando **\seme**). È possibile usare i classici meccanismi di sezionamento di L^AT_EX.

All'interno di questi file gli esercizi a risposta chiusa vengono inseriti nell'ambiente **test** che ha come argomento opzionale il punteggio (un intero positivo). Ogni gruppo di quesiti è inserito nell'ambiente **questions**⁸.

Le altre tipologie di esercizi non hanno invece alcun ambiente particolare.

Il file totale-versioni

Il file **totale-versioni** serve per generare tutte le varianti di un esercizio contenuto in un file ed è usato in fase di costruzione del database degli esercizi con funzione di controllo. Ha sempre l'opzione **prova** (si veda 3.3); la compilazione genera una versione numerica e usando l'opzione **param** la versione parametrica degli esercizi. Al suo interno si deve usare un solo comando, **\esercizio{nome}**, che deve contenere il nome dell'esercizio da compilare. Se si compila la versione parametrica verranno indicati i parametri e i loro intervalli di variazione. Il meccanismo con cui opera il file è simile a quello di **\selectallproblems** del pacchetto **probsoln**.

3.3 Le opzioni del pacchetto

Il pacchetto **esami** presenta alcune opzioni:

8. In un ambiente **test** possono esistere più ambienti **questions**.

- **allowrandomize** e **norandomize**: la prima permette di permutare in modo casuale l'ordine delle risposte nelle domande a risposta chiusa, la seconda di non permutarlo (di default, le risposte vengono permutate in modo casuale);
- **shuffle**, **shufflerandom** e **noshuffle**: la prima permette di permutare in ordine casuale gli esercizi quando il numero è maggiore di 8 e in modo deterministico se minore, la seconda di permutarli casualmente quando il numero è maggiore di 6 e minore altrimenti; la terza di non permutarli (l'opzione di default è **shuffle**)⁹;
- **xxxx**: fa caricare un file di configurazione (**esami-xxxx.cfg**) con ulteriori informazioni relative a questo compito (per esempio, il nome dell'esame e le istruzioni). Per far leggere un file di configurazione senza modificare il file **esami.sty** è sufficiente scrivere un'opzione sconosciuta (ad esempio **zzz**) e scrivere il file **esami-zzz.cfg**. È ovviamente possibile anche modificare il file **esami.sty** in modo da avere maggiore flessibilità nella configurazione;
- **pointsonright**: fa apparire una casella con il punteggio sulla destra dell'esercizio. Se l'opzione non è specificata, la casella non appare;
- **nosolutions**: questa opzione genera il compito senza soluzioni (default);
- **solutions**: genera il file delle soluzioni;
- **solutiononly**: genera un file con le sole soluzioni, senza testo degli esercizi, usando l'ambiente **problema**;
- **prova**: si usa solo con il file **totale-versioni** e serve per visualizzare tutte le varianti di un esercizio. Fa visualizzare automaticamente le risposte esatte negli esercizi a risposta multipla e la soluzione nei problemi da svolgere;
- **param**: serve per visualizzare i nomi dei parametri casuali anziché il valore numerico attribuito. Stampa anche una frase in coda ad ogni esercizio, nella quale sono contenuti il valore iniziale e finale attribuiti ai parametri usati nell'esercizio. Può essere usata solo insieme all'opzione **prova**;
- **correzione**: stampa solo i testi di tutti gli esercizi di un compito, senza soluzioni (funziona solo nel file **totale-versioni**);
- **fillb**: da usare se nel compito ci sono esercizi di tipo **fillin**, per poter inserire le risposte corrette nella stringa delle soluzioni;
- **twocolumns**: per scrivere il testo delle domande a risposta chiusa su due colonne;
- **sansserif**: usa caratteri *sans serif*. Utile soprattutto se si usano le due colonne;
- **autopston** e **autopstoffs**: caricano il pacchetto **auto-pst-pdf** e permettono di compilare il file direttamente con pdfLATEX (anche se contiene codice di **pstricks**). La prima opzione crea e include direttamente nel documento i file delle immagini, la seconda non crea i file delle immagini (da utilizzare quando il documento non contiene immagini perché rende più veloce la compilazione)¹⁰.

3.4 I comandi

I comandi per lavorare con i parametri

Come detto, uno degli obiettivi del pacchetto è l'uso di parametri casuali all'interno degli esercizi. Per i nostri scopi abbiamo definito solo parametri interi, ma nulla vieta di implementare procedure simili per parametri razionali o in forma decimale. I parametri vengono definiti con il comando `\FPsetpar[⟨seme⟩]{⟨nome-parametro⟩}{⟨inf⟩}\{⟨sup⟩}[⟨valori esclusi⟩]`. Il parametro avrà nome `\nome-parametro` e varierà tra `⟨inf⟩` e `⟨sup⟩` (inclusi). Il parametro opzionale `⟨seme⟩` serve a imporre un seme diverso per la generazione del numero casuale. Il valore di default del parametro è `\seme` ed è definito a partire dalla data dell'esame e dal numero della versione. È possibile escludere dalla scelta uno o più valori (parametro `⟨valori esclusi⟩`). Se i valori sono più di uno devono essere posti tra parentesi graffe e separati tra virgole; tra i valori esclusi ci possono essere anche altri parametri purché già definiti in precedenza (anche il primo e l'ultimo valore possono essere parametri: in questo caso bisogna prestare attenzione al fatto che il primo valore deve essere minore dell'ultimo). Il meccanismo usato per soddisfare tali condizioni consiste nel ripetere la generazione più volte: il numero massimo di ripetizioni è determinato dal parametro `\maxLoopLimit`, il cui valore di default è 10, ma che è definibile anche in fase di compilazione.

Ad esempio, i comandi:

```
\FPsetpar{a}{2}{10}[3]
\FPsetpar{b}{4}{12}[{\a,6}]
```

creano due variabili `\a` (che può assumere un valore casuale tra 2 e 10, escluso 3) e `\b` (che può assumere un valore tra 4, 12, escluso 6 e il valore

9. La scelta di 8 esercizi per l'opzione **shuffle** è dovuta al limite di 9 parametri dei comandi LATEX, mentre abbiamo scelto 6 esercizi per l'opzione **shufflerandom** perché in tal caso la probabilità di esercizi simili vicini è per noi accettabile; tale numero può essere modificato commentando o scommentando una riga di codice nel file **esami.sty**.

10. Per usare **auto-pst-pdf** è necessario eseguire pdfLATEX con l'opzione **write18**.

già assegnato ad `\a`). In questo caso, il seme è quello stabilito nel preambolo del documento.

Sui parametri è possibile eseguire operazioni sia usando i comandi del pacchetto `fp` sia utilizzando comandi da noi definiti.

Il comando `\FPsv[⟨cifre decimali⟩]{⟨operazione su parametri⟩}` serve ad eseguire operazioni su parametri e restituirne il valore numerico (con il numero prescelto di cifre decimali) o stampare l'operazione eseguita (se si sceglie l'opzione `param` nel pacchetto `esami`). Ad esempio se $k = 2$: `\FPsv{2*k+1}` produce 5 o $2 * k + 1$ con l'opzione `param`; `\FPsv[2]{(2*k+1)/2}` produce 2.50 o $(2 * k + 1)/2$.

La sintassi delle operazioni è la stessa del pacchetto `fp` e se il pacchetto usa l'opzione `param` ogni operazione viene racchiusa tra parentesi tonde per comodità di lettura.

Il comando `\FPval{⟨nome⟩}{⟨operazione su parametri⟩}` assegna a `⟨nome⟩` il valore dell'operazione arrotondato (è il comando `\FPeval` di `fp` modificato) o stampare l'operazione eseguita se si sceglie l'opzione `param` nel pacchetto `esami`. Ad esempio il codice:

```
\FPsetpar{k}{1}{3}
\FPval{a}{2*k+1}
\FPsetpar{b}{2}{20}[a]
```

genera una variabile `\b` che assume un valore casuale tra 2 e 20, escluso (in questo caso) il valore assegnato a `\a`. Nella versione parametrica appare invece una stringa di questo tipo (per quanto riguarda la variabile `\b`):

Il parametro b varia da 2 a 20. $b \neq (2 * k + 1)$.

Sono stati definiti anche alcuni comandi per la semplificazione di frazioni che possono essere usati anche per la formattazione del testo. Il comando `\sempli{⟨num⟩}{⟨den⟩}` serve per semplificare frazioni in cui il numeratore o il denominatore contengono parametri. Il comando `\sempli{2*k}{3*k+1}` per $k = \frac{1}{2}$ genera $\frac{1}{2}$.

Il comando `\semplix{⟨num⟩}{⟨den⟩}` serve per semplificare frazioni in cui il numeratore o il denominatore contengono parametri, ma nelle quali il risultato pari ad 1 non deve apparire e il risultato -1 deve apparire come $-$ (ad esempio perché davanti ad una incognita). Il comando può essere utilizzato ponendo il denominatore uguale ad 1 anche per la formattazione di coefficienti. Ad esempio, se il parametro k assume valore 2: `\FPsv{k-1}x` fornisce $1x$ mentre `\semplix{k-1}{1}x` dà x .

Il comando `\esempli{⟨num⟩}{⟨den⟩}` serve per semplificare delle frazioni in cui il numeratore o il denominatore contengono parametri, ma nelle quali il risultato pari ad 1 non deve apparire, ma deve apparire esplicitamente il valore -1 (ad esempio per gli esponenti). Il comando può essere utilizzato ponendo il denominatore uguale ad 1 anche

per la formattazione di esponenti. Ad esempio, se il parametro k assume valore 2: `x^{-\FPsv{k-1}}` fornisce x^1 mentre `x^{-\esempli{k-1}{1}}` dà x .

Il comando `\sempliz{⟨num⟩}{⟨den⟩}` serve per semplificare frazioni in cui il numeratore o il denominatore contengono parametri e che possono assumere il valore 0 (ad esempio nelle risposte).¹¹

Il comando `\simsqrt{⟨ind⟩}{⟨rad⟩}` serve per “portar fuori” da una radice tutti i fattori possibili, su tali fattori non è possibile eseguire altre operazioni (semplificazioni, ecc.). Il primo parametro obbligatorio, `⟨ind⟩` è l'indice della radice e può essere un numero o un parametro; il secondo, `⟨rad⟩` il radicando e può essere un numero, un parametro o un'operazione su parametri. Ad esempio con $a = 2$ e $b = 1$, `\simsqrt{2}{a^2+4*b}` fornisce $2\sqrt{2}$.

I comandi per la gestione degli esercizi e delle liste

I comandi per la gestione degli esercizi e delle liste rientrano in un'unica categoria in quanto hanno lo stesso tipo di funzionamento: dato un elenco di nomi, riordinare tale elenco in modo casuale ed estrarre poi un certo numero di elementi.

Il comando fondamentale per la gestione degli esercizi è `\esercizi{⟨file 1⟩, ⟨file 2⟩, ..., ⟨file n⟩}` che sceglie un esercizio casuale da ognuno dei file specificati, riordina casualmente gli n esercizi scelti e li manda in output.

Il comando `\estrai[⟨m⟩]{⟨lista⟩}{⟨nome⟩}` estrae $n - m$ elementi da `⟨lista⟩`; gli elementi estratti avranno nome `\nomei`, `\nomeii`, e via di seguito. Ad esempio con il comando `\estrai[1]{insiemi,logica,potenze}{alpha}` si scelgono 2 elementi dell'insieme e questi due elementi si chiameranno `\alphai` e `\alphaii`. Se questi nomi corrispondono a nomi di file di esercizi è possibile inserire nella lista di file argomento del comando `\esercizi` anche `\alphai` e `\alphaii`.

Il comando `\estraialfa{⟨n⟩}{⟨lista⟩}{⟨nome⟩}` estrae $\langle n \rangle$ oggetti casuali da `⟨lista⟩`, mantenendo l'ordine fissato. Gli elementi estratti si chiameranno `\nomei`, `\nomeii` e via discorrendo. Ad esempio con il comando `\estraialfa[2]{a,b,c,d}{alpha}` si scelgono 2 elementi dell'insieme, mantenendo l'ordine alfabetico e questi due elementi si chiameranno `\alphai` e `\alphaii`.

Infine il comando `\estraies[⟨m⟩]{⟨lista⟩}` funziona come il comando `\estrai`, ma esclusivamente su una lista di esercizi; una volta effettuata l'estrazione, manda gli elementi estratti in output come il comando `\esercizi`.

Tali comandi permettono ulteriori meccanismi aleatori di scelta degli esercizi: ad esempio, se il database contiene 20 esercizi su un dato argomento e se ne vogliono inserire nel compito 4, un comando

11. Con gli altri comandi di semplificazione il risultato 0 dà luogo a un errore e al blocco della compilazione.

permette la scelta casuale dei 4 esercizi dell'insieme da cui poi estrarrà una variante.

È possibile sia inserire tutti gli esercizi con un unico comando, sia usare più comandi `\esercizi` o `\estraies`. Questa possibilità è utile ad esempio se si vuole che ci siano esercizi da due sottoinsiemi diversi (ad esempio 5 esercizi sui limiti scelti tra 7 e 3 esercizi sugli asintoti scelti tra 5) o se si vuole, per esigenze di impaginazione, avere alcuni esercizi su due colonne ed altri su una colonna.

Riferimenti bibliografici

ARSENEAU, D. «Il pacchetto `random`». CTAN:
`/macros/generic/misc/random.tex`.

MEHLICH, M. (1999). «Il pacchetto `fp`». CTAN:
`/macros/latex/contrib/fp`.

MESSINEO, G. e VASSALLO, S. (2012). «L^AT_EX per la creazione di temi d'esame. Seconda parte: esami online». Contributi di ricerca in Econometria e Matematica 125, Università Cattolica del S. Cuore, Milano.

PENNA, S. (2006). «Test interattivi di matematica e fisica on-line: il L^AT_EX come strumento di sviluppo». *ArsT_EXnica*, **2**, pp. 65–70.

PILU, A. (2011). «La creazione di una prova d'esame». *ArsT_EXnica*, **11**, pp. 5–14.

STORY, D. P. (2012). «Exerquiz & AcroT_EX». URL
<http://www.acrotex.net/>.

TALBOT, N. L. (2011). «Il pacchetto `probsoln`». CTAN:`macros/latex/contrib/probsoln`.

- ▷ Grazia Messineo
Università Cattolica Milano – ITC “G. Falcone” Corsico
`grazia dot messineo at unicatt dot it`
- ▷ Salvatore Vassallo
Università Cattolica Milano
`salvatore dot vassallo at unicatt dot it`

Test online di matematica: il Progetto M.In.E.R.Va*

Grazia Messineo, Salvatore Vassallo

Sommario

Vengono presentati i pacchetti **esami-online**, per la creazione di prove d'esame con correzione automatica da somministrare al pc, e **minerva**, per la creazione di percorsi di recupero delle carenze in matematica per studenti delle scuole superiori.

Abstract

We present the packages **esami-online**, to create online exams with automatic corrections, and **minerva**, to create sets of exercises for high school students.

1 L'esame online

1.1 Introduzione

All'Università Cattolica, il progetto di somministrare una parte dell'esame di Matematica Generale in via automatica era nato già negli anni '80 da un'idea del professor Giovanni Melzi ed era stato sviluppato dalla professoressa Monica Bianchi (BIANCHI e MELZI, 1990). I mezzi, estremamente limitati sia per quanto riguarda l'hardware che il software a disposizione, sia per il numero di persone coinvolte nel progetto, non ne hanno permesso uno sviluppo completo.

Nell'A.A. 2008/09 il progetto è stato ripreso e si è pensato di modificare opportunamente il software sviluppato per la preparazione dei temi d'esame, in modo da ottenere test da somministrare online.

1.2 Le altre esperienze

Innanzitutto sono state analizzate le soluzioni adottate in altri contesti, sia utilizzando piattaforme specifiche, sia utilizzando $\text{\LaTeX}$ all'interno di pagine web create dinamicamente con linguaggi specifici (ASP, PHP, ecc.).

Le esperienze più significative e facilmente trasferibili sono sembrate:

- i pacchetti $\text{\LaTeX}$ **exerquiz** e **eqexam** del prof. Story (Università di Akron, Ohio) (STORY, 2012), su cui si basano anche i software presentati di seguito e quello da noi elaborato.
- Il software **MacQTeX** creato alla Macquarie University di Sidney, che attraverso l'uso di

una versione modificata del pacchetto **exerquiz**, di alcuni script in Perl e di un database MySQL, permette la creazione "al volo", tramite un'interfaccia web, di set di esercizi (scelti dal docente). Questo software viene utilizzato per attività di recupero autonomo o di tutorato, e anche per svolgere esercitazioni valutate, che contribuiscono al punteggio finale dell'esame. La piattaforma web consente due tipi di accesso: uno per lo studente, che sceglie quali gruppi di esercizi svolgere tra quelli proposti, e uno per il docente, che crea i set di esercizi da somministrare agli studenti ed esamina le statistiche e le valutazioni ad essi relative (GRIFFIN).

- Il software **ExOmatiK**, sviluppato dal prof. Nicolas Poulain del Collège Mallarmé di Parigi per la scuola superiore, ma adattabile anche per l'università. Questo software permette la creazione al volo di fogli di esercizi, scelti da un database e la cui anteprima è presentata online. Gli esercizi possono contenere anche elementi aleatori. Viene fornito, per ogni set di esercizi creato, anche il file sorgente $\text{\LaTeX}$ (POULAIN).
- Il software **AcroWeb** del professor Robert Mařík dell'Università Mendel di Brno, che usa una versione personalizzata del pacchetto **exerquiz**, alcuni altri pacchetti $\text{\LaTeX}$ e alcuni script in Perl, per consentire la creazione "al volo" di set di esercizi da svolgere online, e Cards di esercizi presentati in modo casuale, e altri strumenti didattici interattivi (MAŘÍK).
- Il software **MCQ-Xe $\text{\LaTeX}$** del prof. Davide Ferrario dell'Università Bicocca di Milano, che usa alcuni pacchetti $\text{\LaTeX}$ e alcuni script in Python per generare set di esercizi che possono essere fruiti in tre modi: a stampa (viene generato un file PDF che può essere stampato, completato dallo studente e letto da un lettore ottico), sul web (viene generato un output in HTML che usa MathJax per la parte matematica) e su una piattaforma dedicata (viene generato un output compatibile con la piattaforma e-learning gratuita Moodle in uso presso l'Università Bicocca) (FERRARIO).

Tutte le soluzioni esaminate sono state scartate a causa di vincoli tecnici che impedivano l'installazione sul server messi a disposizione del software necessario per la compilazione.

*Il lavoro si inserisce nel progetto di ricerca di interesse d'Ateneo dell'Università Cattolica "Progetto M.In.E.R.Va.: Creazione di esercizi di tipo interattivo con percorsi differenziati per il recupero delle carenze e la valorizzazione delle eccellenze in matematica".

Inoltre, come già è avvenuto per la gestione dei temi d'esame tradizionali, si è preferito creare un software basato esclusivamente su $\text{\LaTeX}$ sia per la formattazione del testo, sia, per quanto possibile, per la gestione della parte matematica e degli elementi aleatori presenti negli esercizi.

La scelta è stata quindi quella di adattare il pacchetto già creato per la versione standard degli esami (si veda (MESSINEO e VASSALLO, 2010)) affinché fosse possibile creare quiz (a risposta chiusa) da somministrare agli studenti in modalità online, utilizzando file PDF interattivi.

1.3 L'interfaccia

I quiz che costituiscono la prova d'esame sono presentati allo studente sotto forma di un file PDF reso interattivo con l'uso di codice JavaScript (ADOBE SOLUTIONS NETWORK, 2001).

Lo studente è identificato mediante il numero di matricola: se esiste sul server un file che ha come prima parte del nome quel numero, esso viene proposto allo studente. In tal caso, i campi cognome, nome e numero di matricola sono precompilati (fig. 2).¹

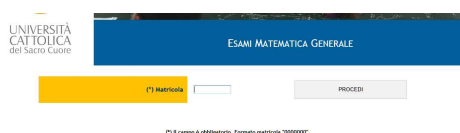


FIGURA 1: La maschera iniziale



FIGURA 2: La maschera di inserimento dati (normalmente già compilata)

Cliccando sull'apposito bottone "Inizio Quiz" lo studente può iniziare la prova. Nell'angolo in basso a sinistra dello schermo appare un segnatempo con avvisi visivi (rettangoli rossi o gialli) che cambiano colore in base al tempo rimanente. A tempo scaduto, lo studente viene avvisato mediante una finestra pop-up. Se, nonostante ciò, invia i dati al server oltre il tempo indicato (più un eventuale bonus) la prova viene annullata; sul server vengono però ugualmente memorizzati i dati.

1. Il sistema permette anche l'utilizzo di matricole "fittizie", cioè non corrispondenti ad alcuno studente per gestire casi eccezionali (studenti privi di matricola, iscrizioni tardive all'esame, ecc.). In tale caso lo studente riempirà i campi corrispondenti ai propri dati.

Lo studente può muoversi all'interno del file utilizzando gli appositi comandi (freccie) poste in basso (fig. 3).

Ogni domanda appare su una sola schermata e prevede una sola risposta esatta. È possibile cambiare una risposta già data, mentre non è possibile annullarla.²

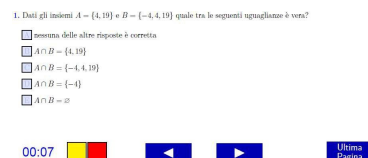


FIGURA 3: Una domanda, i pulsanti di controllo e il contatore con gli alert giallo e rosso

A prova terminata, lo studente clicca sul bottone "Fine Quiz" (fig. 4): tale azione determina l'esecuzione di una serie di istruzioni JavaScript:

- viene valutata la correttezza o meno delle risposte date;
- vengono mostrati allo studente il punteggio ottenuto e l'esito dell'esame;
- vengono inviati al server i dati dello studente, l'orario di inizio e di fine della prova, le risposte date, il punteggio ottenuto.³

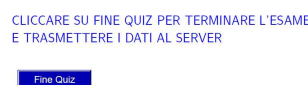


FIGURA 4: La pagina finale

Allo studente viene quindi mostrata un'altra pagina HTML che riepiloga i suoi dati e l'esito. Successivamente il sistema ritorna alla pagina iniziale o presenta allo studente un'altra parte di esame, se previsto.

Lo studente può modificare le risposte finché non ha cliccato su "Fine Quiz": quando lo ha fatto non può più operare alcuna modifica e, se prova comunque a farlo, riceve un messaggio di errore.

1.4 La creazione dei file PDF

Le prove d'esame sono generate usando il linguaggio $\text{\LaTeX}$. La scelta di $\text{\LaTeX}$ rispetto ad altri strumenti è dovuta ad alcuni fattori:

- la qualità necessaria dell'output;
- l'esistenza di un database di esercizi già predisposti (si veda (MESSINEO e VASSALLO, 2010));

2. La cosa non è comunque influente poiché nelle nostre modalità d'esame non sono previste penalizzazioni in caso di risposte errate. È comunque possibile prevedere un'alternativa "non risponde" per tali situazioni.

3. Tali dati vengono memorizzati in un file .csv e/o in un database.

- c) la possibilità di modificare secondo le nostre necessità il software, data la natura delle licenze (GPL o simili);
- d) la possibilità di operare senza installazione di software aggiuntivo sui server dell'università.

A differenza di tutte le altre esperienze a noi note di utilizzo di L^AT_EX per la creazione di quiz online (o anche offline) il software da noi creato non utilizza alcun programma esterno né per la randomizzazione degli esercizi né per la manipolazione delle espressioni matematiche (si veda (MESSINEO e VASSALLO, 2010)).

Anche questo pacchetto, che abbiamo chiamato **esami-online**, si basa su una versione altamente modificata di **exerquiz** di Donald P. Story (STORY, 2012) e prevede anche l'uso dei pacchetti **eq2db**, che consente la comunicazione di stringhe di dati dal file PDF al database sul server, e **insdljs**, che funge da interfaccia tra i comandi L^AT_EX e i comandi JavaScript.

Le principali modifiche (molte delle quali comuni con il pacchetto **esami**) riguardano:

- la completa randomizzazione nella scelta e nella presentazione degli esercizi, che si basa sul numero di matricola dello studente e sulla data dell'esame, minimizzando quindi la possibilità sia di avere versioni simili nella stessa data, sia di riproporre allo stesso studente gli stessi esercizi in date diverse.⁴

Questa scelta è diversa da quella effettuata per gli esami tradizionali (si veda (MESSINEO e VASSALLO, 2010)). Infatti, quando si somministrano le prove in forma cartacea è necessario assicurarsi che versioni contigue della stessa prova siano abbastanza diverse tra loro; quando le prove vengono somministrate online, invece, è impossibile sapere come si disporranno gli studenti. Questa differenza ci ha suggerito di scegliere per la modalità online una gestione totalmente casuale degli elementi aleatori.

- La possibilità di utilizzare esercizi in forma parametrica (come già avviene per gli esami tradizionali).
- La possibilità di utilizzare set di esercizi, scelti in modo casuale, per comporre diverse versioni della stessa prova (il pacchetto **eqexam** di D.P. Story usa esercizi "fissi" e prevedeva, quando abbiamo cominciato a lavorare su questo progetto, la possibilità di creare solo due varianti. Inoltre, gli elementi che devono variare tra una versione e l'altra devono essere modificati "a mano". Il pacchetto **exerquiz** non prevede invece varianti).

4. Il database dal quale vengono scelti gli esercizi viene integrato (aggiungendone di nuovi) solo occasionalmente.

- La modalità di lettura dei dati inviati al server (si veda la sezione 1.5).

La scelta degli esercizi da inserire in ogni prova è stata gestita in due modi:

- a) in una prima fase si è utilizzata la stessa modalità delle prove cartacee: gli esercizi vengono scelti da una persona che si preoccupa di comporre il tema d'esame, di valutare il livello di difficoltà, ecc. In questo caso, la scelta casuale riguarda i valori da attribuire ai parametri, la scelta della variante dell'esercizio, la rotazione delle domande e delle risposte.
- b) In una seconda fase (in particolare quando la modalità è stata estesa ai corsi con un maggior numero di studenti, come descritto nel paragrafo 1.6) si è deciso invece di rendere gli esercizi del database omogenei per livello di difficoltà e di lasciare quindi che la scelta degli esercizi da inserire fosse effettuata direttamente, in modo casuale, in fase di compilazione.⁵

Non potendo installare sui server software aggiuntivo, partendo dall'elenco degli iscritti alla prova abbiamo optato per creare un file PDF per ogni studente; tali file vengono caricati sul server e resi disponibili il giorno dell'esame.

Come già accennato nel paragrafo 1.3, il file viene nominato con il numero di matricola dello studente seguito da un codice identificativo della prova, così che solo gli studenti iscritti possono accedere al file.

Nel processo di compilazione vengono creati anche un file PDF della prova in forma stampabile e un file PDF con le soluzioni: questi file possono essere utili sia durante la discussione della prova con lo studente sia in casi di emergenza (ad esempio problemi sul server).

Per la creazione di questi tre file PDF è stato predisposto un file master, dal nome **master-csv.tex**, che utilizza, nella compilazione, un altro file, dal nome **master.tex**. Compilando il primo file, se ne creano altri, che vengono rinominati con il numero di matricola dello studente e il codice identificativo della prova. Il pacchetto fondamentale utilizzato in questo file è **datatool**, che permette di caricare il database (un file **.csv**) con i dati degli iscritti alla prova.⁶

Il codice usato nel file **master-csv.tex** è

5. In particolare, si è fatto uso del comando **\estraies**, già descritto nell'articolo "Il pacchetto **esami** per la creazione di prove scritte". Prossimamente, sarà sperimentata una modalità diversa, che consentirà di scegliere casualmente e in automatico gli esercizi dall'intero database, specificando solo quanti se ne vogliono per tipo.

6. Nel file **master.tex** viene caricato, ovviamente, anche il pacchetto **esami-online**, le cui opzioni fondamentali sono le stesse del pacchetto **esami** e sono state descritte nell'articolo "Il pacchetto **esami** per la creazione di prove scritte".

```

\usepackage{datatool}
...
\DTLloadddb{Iscritti}
{Elenco_studenti_xxx.csv}
%% il nome del database
\DTLforeach{Iscritti}
{\matricola=Matricola,\nome=Nome,%
\cognome=Cognome}
{\immediate\openout\dati=\matricola.dati
\immediate\write\dati
{\string\def\string\nomeesame{\nomeesame}}
\immediate\write\dati
{\string\def\string\nome{\nome}}
\immediate\write\dati
{\string\def\string\cognome{\cognome}}
\immediate\write\dati
{\string\date{\dataesame}}

\immediate\closeout\dati

```

La seconda parte delle istruzioni serve a creare un file dal nome `matricola.dati`, nel quale sono contenute tutti i dati dello studente (nel nostro caso, cognome, nome, data della prova, nome della prova). Tali informazioni saranno poi usate nella creazione del file PDF, durante la quale verrà inserito automaticamente il nome dello studente (per evitare di dover gestire manualmente nel database i tanti errori compiuti nella digitazione).

Nello stesso file, viene poi eseguita una serie di istruzioni con il comando `\write18` che consentono di compilare, studente per studente, il file per il test cartaceo (con generazione delle eventuali figure), quello per il test online e, infine, quello con la stringa delle soluzioni. La scelta di eseguire istruzioni con il comando `\write18` comporta alcuni problemi di sicurezza e quindi non sarebbe utilizzabile se la compilazione avvenisse “al volo” su un server. È al momento l’unica soluzione che abbiamo trovato per far eseguire, in fase di compilazione, i comandi DOS necessari per creare cartelle, copiare o spostare i file, ecc.

Ad esempio, il seguente codice

```

\immediate\openout
\compilazione=compilazione.txt
\immediate\write\compilazione
{\string\def\string\ncompilazione{1}}
\immediate\closeout\compilazione

\immediate\write18
{copy \nomeesame.tex
\matricola-\modulo.tex}
\immediate\write18
{pdflatex -shell-escape
\matricola-\modulo.tex}

...

\immediate\write18
{move \matricola-\modulo.pdf

```

```

output/\matricola-\exte.pdf}
\immediate\write18
{move \matricola-\modulo.tex output}
...

```

consente di creare una copia del file master rinominata con il numero di matricola dello studente e il codice identificativo della prova, di compilarla, di spostare il file PDF e il file `.tex` rinominato in una cartella chiamata `output` (il file PDF viene anche rinominato durante lo spostamento).

Il file `compilazione.txt` che viene creato all’inizio della procedura, contiene un unico comando, `\ncompilazione`, che assume in sequenza i valori da uno a tre e consente di passare alcune opzioni al pacchetto `esami-online`, che variano a seconda del tipo di output che si vuole ottenere.

I file (sia il pacchetto `esami-online`, sia i file di configurazione) contengono anche un certo numero di istruzioni JavaScript che rendono interattivo il file PDF di output.

Ad esempio, il seguente codice

```

...
\ifthenelse{\equal{\nome}{}}
{\ifeqforpaper
\newcommand\FirstName[2]{\hrulefill}
\else
\newcommand\FirstName[2]
{\textField{IdInfo.Nome}{#1}{#2}}
\fi
\newcommand{\avvertenza}
{\noindent\textcolor{blue}
{\textsf{SCRIVERE NOME E COGNOME
E CLICCARE SU ‘‘INIZIO QUIZ’’}}}
}
{\newcommand\FirstName[2]{\nome}
\addHiddenTextField{IdInfo.Nome}{\nome}
\newcommand{\avvertenza}
{\noindent\textcolor{blue}
{\textsf{VERIFICARE I DATI INSERITI
E CLICCARE SU ‘‘INIZIO QUIZ’’}}}
}
...

```

consente di creare “spazi” diversi per il nome, in relazione al fatto che possa essere letto o no dal file `.csv` e in relazione al fatto che si stia producendo una prova interattiva o una prova cartacea: se la prova è interattiva e il nome non è acquisito dal file `.csv`, compare un campo nel quale si deve inserire il dato da tastiera, mentre se il nome è acquisito viene stampato nel campo corrispondente; se si sta producendo una prova cartacea, nel caso di nome non acquisito viene stampata una linea.

La prova somministrata allo studente può essere singola oppure divisa in più parti. In quest’ultimo caso, anche questo aspetto è gestito da comandi JavaScript, che consentono di indirizzare lo studente verso la parte successiva della prova, oppure di far-

lo uscire dalla procedura, sulla base del punteggio ottenuto.

1.5 La trasmissione dei dati e l'elaborazione lato server

L'applicazione online implementata per la gestione degli esami di Matematica Generale risiede su una partizione dedicata di un server che utilizza come web server Microsoft IIS e le pagine web sono scritte in linguaggio ASPX.

La piattaforma prevede due tipi di accessi:

- *Per i docenti:* è stata predisposta una sezione dedicata all'amministrazione dei test, a cui è possibile accedere solo come utenti autorizzati (sono previsti diversi livelli di autorizzazione). Mediante l'interfaccia è possibile compiere varie operazioni amministrative (modificare la visibilità del sistema, compiere verifiche, stabilire l'ordine in cui si devono svolgere i quiz nel caso di prova suddivisa in più parti, visualizzare, stampare ed archiviare gli elenchi degli esiti ed altro ancora).
- *Per gli studenti:* l'accesso avviene inserendo il numero di matricola e una password casuale generata dal sistema, come spiegato nel paragrafo 1.3. Il software cerca quindi se sul server sia presente un file che abbia come prima parte del nome il numero di matricola inserito. Tale file viene immediatamente rinominato per motivi di sicurezza.⁷ Lo studente svolge l'esame. Al termine, l'esito dell'esame viene inviato al sistema e salvato. Infine, viene disabilitato l'accesso dello studente, forzandone il logout.

La fase più delicata di tutto il processo è risultata essere quella finale. Infatti quando al termine della prova lo studente clicca su **Fine Quiz**, il codice JavaScript contenuto nel file PDF elabora le risposte, comunica mediante un apposito form allo studente il punteggio e l'esito e trasmette al server un file FDF (*Forms Data Format*) (ADOBE SOLUTIONS NETWORK, 1999) che contiene tutte le informazioni necessarie (dati dello studente, dati della prova, risposte date, esito, orario di inizio e fine della prova).

Nelle prime prove, i contenuti del file FDF venivano interpretati da uno script ASP e salvati sul server in un file in formato `.csv` che poteva essere scaricato dai docenti alla fine della prova. Questa soluzione aveva il vantaggio di un'enorme semplicità sia nell'implementazione che nell'utilizzo (lo script ASP è fornito insieme al pacchetto *exerquiz* e richiede una personalizzazione minima; il file `.csv` può essere scaricato e letto su qualsiasi macchina

7. Per esempio, impedire che un altro studente possa accedere da altra posizione allo stesso file per eseguire la prova al posto del candidato, o situazioni simili.

e sistema operativo e non necessita di particolari applicativi per la sua creazione e compilazione).

All'aumentare del numero di studenti coinvolti questa soluzione ha mostrato diversi problemi, in particolare nella scrittura dei dati nel caso di accessi contemporanei, poiché i dati potevano essere registrati con un certo ritardo.

Per questo motivo, nelle prove successive si è deciso di cambiare la modalità di registrazione dei dati che ora avviene in un database MSSql Server.

Il file FDF viene analizzato da uno script (attualmente in linguaggio ASPX) che provvede a scrivere i dati sia in un database che nel file `.csv` (che serve ora solo da backup). Dal database vengono generati automaticamente gli elenchi degli esiti suddivisi per gruppo e per data di svolgimento della prova ed è possibile ovviamente utilizzarlo anche per la ricerca di esiti relativi a un singolo studente, ecc.

Uno dei problemi che sembrava essere di difficile soluzione in fase di progettazione era quello della riproducibilità e conservazione della prova come svolta dallo studente per il periodo stabilito dalle norme. La prima soluzione adottata è stata conservare nel file `.csv` e nel database la stringa delle risposte data dallo studente, i suoi dati e tempi di esecuzione, unitamente al file PDF della sua prova (non compilato). Inoltre la modalità di creazione del file consente di ricrearlo identico se necessario anche a distanza di tempo. Un'altra e più efficace soluzione consiste nel memorizzare la coppia file PDF, file FDF in modo che venga salvato oltre al testo della prova anche l'insieme dei campi compilati dallo studente: quindi è possibile ricreare la prova come eseguita dallo studente importando i dati del modulo FDF o aprendolo direttamente con Acrobat Reader.

1.6 La sperimentazione

La prima somministrazione di una prova online è stata effettuata sugli studenti di un corso di Laurea in Economia dell'Università Cattolica il cui numero ridotto era ideale per testare il software.

È stata effettuata una simulazione, nel corso della quale si sono rilevate alcune criticità (layout e impaginazione non ottimali, necessità di un timer che indicasse il tempo residuo) risolte prima della sperimentazione vera e propria.

Gli esiti degli studenti nella prima parte (prova automatica) non si sono discostati significativamente da quelli degli altri gruppi di studenti dei corsi paralleli di Matematica Generale che hanno svolto, nella stessa data, una prova di pari contenuto e difficoltà.

Un questionario di gradimento (anonimo) è stato somministrato agli studenti una decina di giorni dopo la prova e ha mostrato che gli studenti coinvolti hanno apprezzato tale modalità d'esame e soprattutto la possibilità di conoscere immediatamente l'esito ottenuto.

I risultati del questionario sono i seguenti:

- il 51% degli studenti preferisce la modalità on-line rispetto a quella cartacea e il 19% è indifferente;
- il 51% degli studenti preferirebbe sostenere l'intera prova al PC e circa il 7% è indifferente;
- il 91% degli studenti ritiene molto positivo conoscere immediatamente il risultato della prima parte;
- gli aspetti tecnici della prova (interfaccia grafica, tempo a disposizione, chiarezza delle istruzioni, modalità di preparazione) sono stati giudicati positivamente in media da oltre il 70% degli studenti.

L'esito largamente positivo della sperimentazione, protratta per un intero anno accademico, sia dal punto di vista tecnologico che didattico, ha suggerito la possibilità di estendere questa modalità ad una parte dell'esame per tutti i corsi di Matematica Generale, coinvolgendo in alcuni appelli più di 1500 studenti.

1.7 Vantaggi e problemi delle prove automatizzate

La somministrazione delle prove di Matematica Generale con modalità in parte automatizzate presenta diversi vantaggi che possono essere così riassunti:

- il vantaggio principale per i docenti è quello dell'elaborazione automatica dei risultati che, considerando il numero complessivo di prove da correggere, è sicuramente considerevole;
- la votazione della prova (almeno della parte svolta al PC) è immediatamente disponibile per lo studente e, di conseguenza, i tempi di correzione della parte finale dell'elaborato (e quindi di attesa dell'esito per gli studenti) sono molto più brevi;
- la correzione automatica è priva di errori umani che sono sempre possibili valutando un numero molto elevato di prove;
- poiché la correzione automatica è oggettiva e spersonalizzata, dal punto di vista psicologico genera una minore ansia nello studente che riesce ad accettare con maggior serenità l'eventuale esito negativo, evitando inoltre le strumentali polemiche sulla maggiore o minore "bontà" di un docente rispetto ad un altro;
- il fatto che ogni studente abbia un compito diverso e che ogni studente visualizzi al computer solo un esercizio per volta diminuisce il rischio di "comunicazioni" tra studenti;

- la modalità è più facilmente accessibile anche agli studenti diversamente abili (ad esempio dislessici).

I principali problemi della somministrazione delle prove al PC sono invece:

- l'interattività di cui si dispone con i file PDF è limitata, anche se sufficiente per gli scopi attuali (una maggiore interattività potrebbe essere sviluppata usando ad esempio piattaforme di e-learning specializzate nella matematica);
- la messa a punto in anticipo di tutti i temi d'esame pone alcuni problemi di sicurezza che potrebbero essere agevolmente superati permettendo la creazione delle prove "al volo", ossia solo al momento della loro erogazione allo studente;
- la soluzione attualmente implementata non permette la possibilità di utilizzare *data set* di esercizi con la stessa struttura e contenuti di pari difficoltà, come prove di autovalutazione che gli studenti potrebbero svolgere a casa.

2 Il progetto M.In.E.R. Va.

2.1 Introduzione

Il progetto M.In.E.R. Va. (Matematica INterattiva Esercizi di Recupero e VALorizzazione)⁸ è un'evoluzione della gestione informatizzata delle prove d'esame e ha l'obiettivo di fornire uno strumento informatico flessibile che permetta agli insegnanti la creazione di percorsi, anche personalizzati, volti al recupero di debiti formativi o alla valorizzazione delle eccellenze.

Nella scuola secondaria, infatti, l'elevato numero di debiti formativi fatti rilevare dagli studenti in matematica — come le statistiche OCSE-PISA evidenziano — impone ai docenti di predisporre adeguati strumenti per il recupero.

Tradizionalmente, tali debiti vengono recuperati mediante corsi di recupero e assegnazione di compiti (personalizzati) a ciascuno studente. Se il numero di coloro che hanno carenze è elevato, si pone il problema della correzione dei compiti assegnati, spesso molto onerosa. Inoltre, nella maggior parte dei casi il recupero delle carenze passa anche per la ripetizione di molti esercizi tutti simili per acquisire automatismi di calcolo e risoluzione.

I test a risposta multipla con correzione automatica, nei quali vengono utilizzati esercizi scritti in forma parametrica, si affiancano agli interventi didattici tradizionali e mettono lo studente al centro del percorso didattico, permettendogli una maggiore autonomia nei tempi e nei modi di fruizione dei contenuti.

8. Il progetto si inquadra nell'ambito di una ricerca della durata di tre anni dell'Università Cattolica del Sacro Cuore.

Con lo stesso strumento, è possibile proporre percorsi di eccellenza per gli studenti interessati allo studio della matematica, anche in vista di una loro futura iscrizione a facoltà in cui la matematica è oggetto di studio.⁹

Esistono già software che propongono percorsi teorici e pratici utilizzati sia per il recupero del debito nelle scuole secondarie, sia come percorso in molte sedi universitarie.

Il software allo studio si differenzia da quelli già presenti sul mercato per i seguenti motivi:

- 1) la possibilità di fruire autonomamente offline del percorso;
- 2) un'ampia tipologia di esercizi oltre agli usuali test con domande a risposta chiusa singola: risposta aperta (con risposta numerica, vettoriale o in formato testo, anche matematico), completamento, risposta chiusa multipla;
- 3) la possibilità di fruizione senza installare alcun applicativo aggiuntivo se non pochi file su un server per la modalità online;
- 4) un'ottima resa del testo matematico (che è la parte nella quale molti altri software risultano carenti, poiché spesso le formule matematiche vengono importate come immagini, appesantendo la pagina e talvolta compromettendone la visualizzazione);
- 5) indipendenza dal sistema operativo (almeno dal lato utente).

Si è scelto di creare uno strumento informatico indipendente dal sistema operativo usato poiché molte scuole secondarie superiori si stanno orientando, come prevede la normativa, verso sistemi operativi *open source*. Questo aspetto permette da un lato di modificare liberamente il codice necessario, dall'altro di non avere spese per licenze, ma solo (eventualmente) per il materiale.

I docenti delle scuole coinvolte collaboreranno al progetto per quanto concerne la valutazione dei contenuti da proporre (sia in termini di efficacia didattica che in termini di difficoltà) la creazione del database di esercizi e dei percorsi teorici, la creazione e la gestione del supporto informatico per quanto di competenza della scuola, la sperimentazione degli strumenti proposti nelle classi.

2.2 La struttura del progetto

Il progetto prevede la realizzazione di quattro tipologie di contenuti:

- 1) schemi teorici per il rafforzamento o la creazione delle conoscenze;

9. Tali percorsi, pur previsti nel Piano dell'Offerta Formativa delle scuole, sono spesso di difficile realizzazione se gli studenti in questione sono inseriti in gruppi classe numerosi e disomogenei, nei quali le esigenze degli studenti in difficoltà prevalgono.

- 2) esercizi guidati per illustrare le principali modalità di risoluzione dei problemi;
- 3) esercizi di consolidamento e autovalutazione;
- 4) esercizi di verifica dei risultati ottenuti.

I contenuti possono presentare vari livelli di interattività e possono essere fruiti sia offline che online (tutti, tranne gli esercizi di verifica dei risultati ottenuti, che prevedono il tracciamento degli esiti da parte del docente).

Svolgendo gli esercizi in modalità offline, lo studente può ricevere automaticamente, a scelta del docente, i seguenti feedback:

- 1) attribuzione di un punteggio (valutazione);
- 2) segnalazione automatica della risposta esatta/sbagliata;
- 3) suggerimento per arrivare alla soluzione;
- 4) riferimenti ai contenuti teorici;
- 5) schema risolutivo.

Inoltre, nella modalità offline la fruizione dei contenuti avviene senza tracciare l'attività svolta dallo studente (come per le simulazioni delle prove ECDL). Ciò consente di minimizzare l'ansia da prestazione dello studente che può esercitarsi senza sentirsi valutato.

In modalità online è previsto, oltre a quanto già detto, anche il tracciamento, mediante la registrazione su un file che può restare sul server o essere inviato per email al docente e/o allo studente.

2.3 Il pacchetto minerva

Per la realizzazione del progetto, abbiamo predisposto un nuovo pacchetto che abbiamo chiamato *minerva*, mediante il quale è possibile realizzare gli esercizi, sia di consolidamento e autovalutazione, sia di verifica dei risultati ottenuti.¹⁰

Il pacchetto si caratterizza per un'estrema flessibilità, poiché, dallo stesso file di esercizi, consente di ottenere output differenziati che l'insegnante può offrire consequenzialmente o modularmente:

- 1) esercizio autonomo (con eventuale autovalutazione);
- 2) esercizio con monitoraggio degli esiti da parte del docente;
- 3) prova di accertamento.

10. Gli schemi teorici sono realizzati mediante presentazioni (usando *beamer* o strumenti simili) e dispense; gli esercizi svolti sono stati realizzati usando i pacchetti *slidyxx* e *mat-mak* del prof. Mařík, che ci ha messo gentilmente a disposizione anche tutto il materiale da lui predisposto.

Il pacchetto *minerva* può essere considerato un'evoluzione del pacchetto *esami-online*, di cui si è parlato nel paragrafo 1.4.

Le principali somiglianze riguardano il database di esercizi, che viene gestito in modo analogo, la presenza di un file master, dal quale vengono generati tutti gli output necessari per l'esecuzione della prova (file interattivo, file cartaceo, stringa delle soluzioni), i comandi matematici di semplificazione delle frazioni, degli esponenti, ecc. (già condivisi con il pacchetto *esami*). Inoltre, anche questo pacchetto permette di inserire parametri casuali negli esercizi: tale modalità consente allo studente con carenze di ripetere esercizi simili, ma con dati differenti, e di rafforzare la comprensione dei contenuti proposti.

Le principali differenze riguardano:

- le opzioni **compito** ed **esercitazione**: permettono di scegliere se si desidera costruire una prova di verifica vera e propria oppure un'esercitazione, che può essere tracciata oppure da svolgere offline. La principale differenza tra le due modalità consiste nel fatto che nella modalità **esercitazione**, oltre alla segnalazione della risposta giusta o sbagliata a test finito, è possibile inserire anche una traccia di soluzione che consenta allo studente di comprendere in modo autonomo i propri errori. Tale possibilità non è invece prevista nella modalità **compito**, nella quale si può scegliere se indicare allo studente quali risposte sono corrette e quali no. Inoltre, in modalità **esercitazione**, si può scegliere se somministrare a tutti gli studenti lo stesso set di esercizi (in questo caso il file non riporta all'inizio nome, cognome e classe dello studente, ma dei campi vuoti che di volta in volta devono essere compilati) oppure creare set personalizzati (in cui i campi nome, cognome e classe sono precompilati e gli esercizi vengono scelti e ruotati in modo casuale in base al compito che viene compilato).
- L'introduzione di un ambiente **solution** anche per i test a risposta multipla, come si diceva al punto precedente. Questa possibilità era già prevista nel pacchetto *exerquiz*, ma nel nostro caso comportava alcune difficoltà aggiuntive. Infatti, nel pacchetto *exerquiz* il contenuto dell'ambiente **solution** viene scritto *verbatim* su un file esterno, per poter poi essere messo in fondo a tutti i quiz ed essere reso visibile solo al termine della prova. Nel nostro caso ciò era impossibile perché il testo dell'esercizio e della sua soluzione è contenuto nel comando `\newproblem`, che, a nostra conoscenza, non poteva contenere *verbatim* come argomento. Questo problema non si presentava per gli esami cartacei, essendo stato gestito in modo

diverso: infatti il testo della soluzione viene scritto di seguito alla domanda a cui si riferisce se viene data l'opzione **solution** e invece viene cancellato se è data l'opzione **nosolution**. Nel pacchetto *minerva*, il problema dell'ambiente **solution** è stato risolto mediante l'uso del pacchetto *cprotect*.

- Il modo in cui vengono scelti gli esercizi all'interno del database: nei pacchetti *esami* ed *esami-online* gli esercizi vengono scritti ognuno in un file che contiene le n varianti (omogenee) dello stesso problema. L'inserimento di un esercizio in una prova viene fatto scegliendo una variante a caso da ogni file che si decide di inserire. Per le finalità del progetto, era invece opportuno che da uno stesso file si potessero scegliere più varianti dello stesso esercizio (si pensi per esempio ad una esercitazione sulle equazioni di secondo grado, nella quale l'apprendimento passa anche per la ripetizione di molti esercizi tutti simili). A tale scopo, è stato costruito l'ambiente **MNdb**, che ha un solo argomento obbligatorio, ovvero il nome di un file nel quale saranno scritti gli esercizi. All'interno di questo ambiente, il comando `\selectrandomlyn` permette di selezionare n varianti da un file (simile a quello costruito per gli esami tradizionali) e di scriverle nel file esterno definito nell'ambiente. La chiusura dell'ambiente fa scrivere, in ordine casuale o fissato, gli esercizi selezionati.

L'idea di base è quella di creare un database di esercizi che si adattino a diverse tipologie di necessità (esercizi autonomi, prove di valutazione, esercizi guidati). Il database sarà creato con la collaborazione dei docenti delle scuole coinvolte.

2.4 La sperimentazione effettuata nell'a. s. 2011-2012

La sperimentazione delle attività del progetto è iniziata nel settembre 2011. Sono stati coinvolti, nella prima fase, circa 80 studenti di due scuole secondarie di secondo grado, l'ITC "Giovanni Falcone" di Corsico e l'ITI "Giuseppe Lagrange" di Milano.

Nel corso del primo anno di attività del progetto è stato creato il pacchetto *minerva* e i file master necessari per la creazione delle prove: in particolare può essere considerata conclusa sia da un punto di vista "tecnico" che da un punto di vista "grafico" la parte relativa alle prove di esercitazione tracciata (ovvero con esiti salvati su server), di esercitazione autonoma non guidata (esercizi interattivi) e di verifica online con domande a risposta chiusa, valutata in modo automatico. Restano da completare le parti riguardanti le esercitazioni guidate (che vorremmo integrare nel pacchetto *minerva*) e la possibilità di utilizzare domande a risposta aperta o a risposta chiusa multipla.

Dal punto di vista didattico si è cominciato a sviluppare il percorso di recupero e consolidamento per le classi del triennio ITC, per le quali sono stati predisposti i materiali teorici e le esercitazioni (sia tracciate che non) su alcuni argomenti del programma. Parte del materiale predisposto è visibile sul sito web del progetto, all'indirizzo http://www.itcfalcone.it/portale/?page_id=4418.

I dati raccolti nella prima fase della sperimentazione sono in corso di elaborazione.

In alcune classi si è voluta anche sperimentare una modalità didattica alternativa: una parte degli esercizi e del materiale teorico è stata creata da alcuni studenti.

Questa sperimentazione ha dato esito largamente positivo in quanto ha permesso, tra l'altro, una riflessione da parte degli studenti sugli errori più comuni e ha dato la possibilità di far gestire loro, operativamente, problemi parametrici che di solito sono trattati in modo superficiale nello svolgimento "normale" del programma.

Il percorso ha permesso anche lo studio, seppure parziale, del linguaggio L^AT_EX, in collaborazione con i docenti di Laboratorio di Informatica.

L'utilizzo dei materiali prodotti ha reso necessaria la creazione di una piattaforma web attualmente ospitata sul sito dell'ITC "Falcone" e visibile all'indirizzo http://www.itcfalcone.it/portale/?page_id=4418. Viene utilizzato come web server Microsoft IIS. Le pagine sono scritte in linguaggio ASP.

La piattaforma prevede tre tipi di accesso:

- come utente *studente*: questo utente può partecipare agli eventi (prove, lezioni, ecc.) a cui è stato invitato dal docente. In futuro è previsto che possa visualizzare tutti i dati relativi alle sue prove (data di esecuzione, punteggio ottenuto, tempo di svolgimento, ecc.) e anche statistiche anonime relative alle prestazioni di studenti della sua classe o del suo gruppo di lavoro.
- Come utente *docente*: al docente vengono assegnate delle classi o dei gruppi; può creare eventi (prove, lezioni, ecc.) a cui invitare tutti gli studenti della classe/gruppo o solo alcuni; può visualizzare i risultati ottenuti da ciascun gruppo. In futuro potrà accedere a statistiche anonime relative alle prestazioni di tutti gli studenti della scuola. Inoltre, si prevede di integrare nella piattaforma un editor L^AT_EX che funzioni online, per consentire ai docenti la creazione di nuovi esercizi o la personalizzazione di quelli già esistenti. Sono stati valutati a tale scopo alcuni editor *open source*, il cui codice viene ora studiato per consentire la personalizzazione dell'interfaccia e l'integrazione nella piattaforma.

- Come utente *amministratore*: questo utente può creare eventi per tutti gli utenti della piattaforma; può visualizzare i dati relativi ad ogni utente; può compiere tutte le operazioni di amministrazione.

In futuro si prevede inoltre di migrare la piattaforma da server Microsoft IIS a server Apache (è necessario tradurre i file di comunicazione client-server dal linguaggio ASP al linguaggio PHP) e di creare un'interfaccia che consenta di gestire online il database degli esercizi, che sia simile alla piattaforma ExOmatiK (POULAIN).

Riferimenti bibliografici

- ADOBE SOLUTIONS NETWORK (1999). *FDF Toolkit Overview Technical Note # 5194*. ADOBE SYSTEMS INC.
- (2001). *Acrobat Forms JavaScript Object Specification, Version 5.0.5 Technical Note # 5186*. ADOBE SYSTEMS INC.
- BIANCHI, M. e MELZI, G. (1990). «Paut: un programma di autovalutazione». *Didattica delle Scienze*, n. 150.
- FERRARIO, D. «Mcq-xeL^AT_EX». URL <http://www.matapp.unimib.it/~ferrario/main.html>.
- GRIFFIN, F. «MacqT_EX». URL <http://rutherglen.ics.mq.edu.au/~macqtex/>.
- MAŘÍK, R. «Acroweb». URL <http://user.mendelu.cz/marik/index.php?item=41>.
- MESSINEO, G. e VASSALLO, S. (2010). «L^AT_EX per la creazione di temi d'esame. prima parte: la prova scritta tradizionale.» *Contributi di ricerca in Econometria e Matematica 113*, Università Cattolica del S. Cuore, Milano.
- POULAIN, N. «Exomatik». URL <http://www.exomatik.net/>.
- STORY, D. P. (2012). «Exerquiz & AcroT_EX». URL <http://www.acrotex.net/>.

- ▷ Grazia Messineo
Università Cattolica Milano – ITC "G. Falcone" Corsico
`grazia dot messineo at unicatt dot it`
- ▷ Salvatore Vassallo
Università Cattolica Milano
`salvatore dot vassallo at unicatt dot it`

Stemma codicum con L^AT_EX

Matteo Fadini

Sommario

Nell’allestimento di una edizione critica arriva il momento di disegnare lo *stemma codicum*, vale a dire la rappresentazione grafica, tramite un albero genealogico, dei rapporti di derivazione dei manoscritti e delle stampe che trasmettono il testo oggetto dell’edizione.

Non esiste un pacchetto pensato per la creazione degli *stemma*, ma è possibile utilizzarne alcuni normalmente impiegati negli studi di linguistica generativa per gli alberi sintattici, adattandoli allo scopo.

Esistono molti pacchetti di questo tipo, ma si è deciso di limitare la descrizione a due casi: uno di facile utilizzo (*syntree*) e uno più complesso ma molto più versatile (*xytree*). La descrizione dei due pacchetti si limiterà alle funzioni utili per la costruzione degli *stemma codicum*, tralasciando le altre particolarità non pertinenti.

Abstract

During a critical edition’s work it is necessary to draw the *stemma codicum*, i.e. the graphical representation of the lineage relationships of the textual transmission through the linkage of manuscripts and prints.

There is no specific package designed to draw the *stemma*. Nevertheless it is possible to adapt the packages usually employed for generative syntax trees in linguistic studies.

This paper discusses two cases: *syntree*, a user-friendly package, and *xytree*, a more complex yet more powerful package. The analysis limits the description to the useful functions for drawing *stemma codicum*, neglecting the others.

1 Filologia, Lachmann e lo *stemma codicum*

La filologia esiste da quando esiste un interesse culturale per il testo letterario, per la sua interpretazione, comprensione e diffusione *latu sensu* controllata.

Limitandoci ad alcuni casi, si può citare l’edizione dei poemi omerici fatta approntare da Pisistrato nel VI secolo a.C., oppure l’intenso lavoro dedicato dai filologi alessandrini ai testi della letteratura greca. Quasi ogni epoca ha sentito il bisogno di organizzare la produzione letteraria contemporanea e di conservare i testi percepiti come classici.

L’interesse per la correttezza dei testi ritenuti importanti, gli sforzi per rendere comprensibile — tramite commenti, glosse e chiose — il contenuto di queste opere e, infine, la trasmissione delle stesse è ciò che in ogni epoca ha caratterizzato gli studi filologici.

Alcuni termini riguardanti il mondo del libro e ormai di uso comune hanno una derivazione dal lessico specialistico della filologia; per fare alcuni esempi: volume (letteralmente il rotolo di papiro o di pergamena su cui erano scritti i testi; in caso di testi molto estesi occorre diversi *volumina* per trascrivere una sola opera), rubrica (titolo di colore rosso), tomo (letteralmente “taglio”, parte dell’opera contenuta in una pergamena o papiro) e stile (da *stilo*, strumento appuntito per scrivere).¹

Il problema fondamentale con cui si sono dovuti confrontare i filologi di ogni tempo, soprattutto per i testi composti prima della nascita della stampa a caratteri mobili, è la pluralità di manoscritti che tramandano ogni singola opera. Ogni manoscritto di una particolare opera, fatalmente, presenta delle diversità rispetto agli altri: un copista, per attento e scrupoloso che sia, commette degli errori durante il lavoro di copiatura a causa di fraintendimenti di lettura o di lapsus involontari.² Inoltre un manoscritto per così dire *corretto* può, nel corso degli anni, subire dei danneggiamenti: una pagina può staccarsi e perdersi, una macchia (di inchiostro o di umidità) può impedire la lettura di una porzione del testo oppure durante la sostituzione della rilegatura divenuta logora può accadere che si ricomponga in maniera errata il manoscritto. Ogni copia, per quanto corretta, effettuata successivamente a questi accidenti darà come risultato un testo differente rispetto alla forma originaria. Infine possono aver luogo interventi consci e volontari che mirano alla modifica di un testo, per ragioni culturali, ideologiche o religiose.³

I testi dell’antichità non ci sono giunti in *originale*, ma in numerose copie effettuate in età diverse; sicuramente queste copie contengono errori, inversioni e lacune.

Come fare, quindi, per riuscire a leggere il testo, poniamo, dell’*Iliade* o della *Divina Commedia* così come è stato scritto, oppure, il più simile possibile all’originale?

Fino alla rifondazione scientifica della filologia, le procedure filologiche non hanno subito sostan-

1. Si rimanda a MANIACI (2002) per una trattazione organica di questi argomenti.

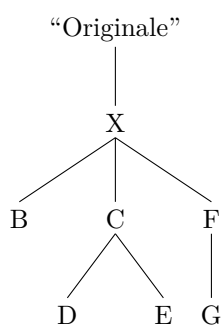
2. A riguardo, si veda DAIN (1975).

3. Per gli approfondimenti, si rimanda a STUSSI (1994).

ziali modifiche rispetto a quelle impiegate dagli alessandrini: il filologo cercava di confrontare (colazionare) i manoscritti di una particolare opera e poi sceglieva tra le varianti, scartando gli errori e le locuzioni ritenute non conformi alla lingua dell'autore e scegliendo le varianti "migliori". Questo tipo di procedimento si basa sul giudizio (e sulla genialità) del filologo, senza nessun metodo che potesse garantire il risultato finale.

Alla figura di Karl Lachmann (1793–1851) è legata la rivoluzione positivista della filologia. Il "metodo di Lachmann"⁴ è presto sintetizzato: si confrontano tutti i manoscritti che trasmettono un particolare testo; sulla base degli errori comuni a più manoscritti si individuano delle *famiglie*, vale a dire dei gruppi di manoscritti legati da rapporti reciproci; infine si scelgono le varianti di per sé indifferenti (del tipo: *sguardo lieto* / *sguardo sereno*, ovviamente non in rima) sulla base della maggioranza tra le famiglie ricostruite. Tutto ciò discende da alcuni assunti, il primo dei quali postula che è molto improbabile che lo stesso errore significativo, non ovvio né facilmente correggibile sia stato commesso *indipendentemente* da due copisti; se due manoscritti hanno quel medesimo errore allora o sono una copia dell'altro o derivano entrambi da un antecedente comune. In ogni caso, questi due manoscritti fanno parte della medesima famiglia e "contano" come uno solo nella scelta delle varianti non significative. Secondariamente si dà per scontato che la variante non significativa presente nella maggioranza delle famiglie abbia più possibilità di essere quella effettivamente scritta dall'autore.

Al termine dell'analisi degli errori, il filologo disegna lo *stemma codicum* che è la rappresentazione grafica dei rapporti genealogici tra manoscritti che è possibile ipotizzare sulla base dell'analisi degli errori comuni.

FIGURA 1: Esempio di *stemma*

Nello *stemma* mostrato nella figura 1 i 6 manoscritti che trasmettono un testo hanno i seguenti rapporti:

1. tutti i manoscritti hanno un errore non banale, e tutti hanno almeno un altro errore

4. Per una presentazione approfondita, si rimanda a TIMPANARO (1963).

significativo, diverso però da manoscritto a manoscritto. Si può dedurre quindi che tutti siano stati copiati da un manoscritto (X), ora perduto, che conteneva quell'errore.

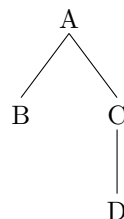
2. G ha tutti gli errori significativi di F ed in più ne ha uno particolare: quindi è copia di F.
3. D ed E hanno tutti gli errori di C ed in più ne hanno, ciascuno, uno proprio: sono quindi due copie indipendenti di C.

Dato quello *stemma*, una variante non erronea comune a C, D ed E "conta" come una variante di B; mentre una variante che sia uguale in B ed F ha buone probabilità di essere d'autore.

2 Pacchetto syntree

Syntree⁵ è probabilmente il pacchetto d'uso più semplice per la creazione di *stemma codicum*: non pone particolari limiti circa il numero di rami né la profondità. Dopo aver inserito nel preambolo la chiamata (`\usepackage{syntree}`), gli stemmi sono creati come segue:

```
\syntree[A [B] [C [D] ] ]
```

FIGURA 2: Esempio di *stemma codicum* semplice (**syntree**)

La sintassi è dunque la seguente: il comando **\syntree** introduce lo *stemma*, racchiuso tra parentesi quadre; il nodo principale è il primo carattere, i nodi secondari sono nuovamente racchiusi tra parentesi quadre.

In alternativa, per evitare di "perdersi" nella scrittura del codice, si può utilizzare la seguente sintassi, del tutto equivalente alla prima:

```
\syntree
[A
  [B]
  [C
    [D] ]
]
```

Ovviamente si può inserire del testo anche maggiore di una lettera, così come del testo formattato come esemplificato dalla figura 3 nella pagina successiva.

Se un nodo è preceduto dai caratteri `.t` il contenuto testuale del nodo sarà collegato con

5. Questo pacchetto è consigliato in LEAL e PIGNALBERI (2012b); per una esauriente trattazione in italiano si veda PIGNALBERI e BINI (2008).

```
\synttree
[\textit{Primo nodo}]
[Primo figlio]
[Secondo figlio [Ultimo ramo] ]
]
```

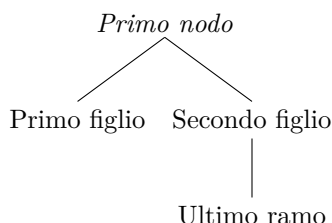


FIGURA 3: *Stemma con testo formattato (synttree)*

un triangolo:⁶ si veda lo *stemma* mostrato nella figura 4.

```
\synttree
[Primo nodo]
[Primo figlio]
[Secondo figlio [.t Ultimo ramo] ]
]
```

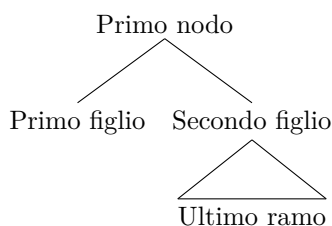


FIGURA 4: Esempio di collegamento a triangolo (*synttree*)

Per assegnare ad un ramo figlio l'ultima posizione, basta far precedere da `.b` il nodo e dichiarare la profondità dell'albero tramite un numero tra parentesi graffe, come si nota nella figura 5.

```
\synttree{3}
[Primo nodo]
[.b Primo figlio]
[Secondo figlio [Ultimo ramo] ]
]
```

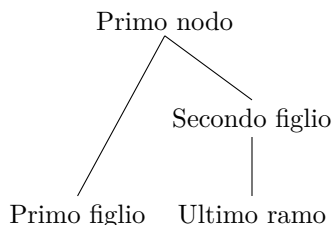


FIGURA 5: *Stemma con collegamento all'ultimo livello (synttree)*

Il comando `.x`, da premettere a un ramo, crea una congiunzione a triangolo al più basso gra-

6. Questa eventualità, a dire il vero poco comune, può essere utilizzata in caso non si voglia o non si possa fornire una spiegazione dei rapporti genealogici tra alcuni testimoni derivanti da un antecedente comune.

do dell'albero (occorre specificare il numero della profondità come sopra); si veda la figura 6.

```
\synttree{4}
[A]
[B [.x Testo] ]
[C [D] [E [G]] [F] ]
]
```

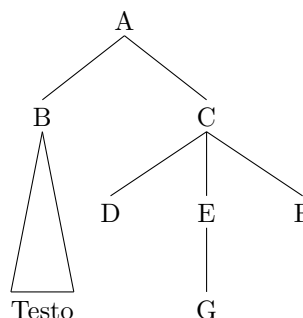


FIGURA 6: *Stemma con collegamento a triangolo all'ultimo livello (synttree)*

Ci sono quattro comandi per modificare la struttura dell'albero, e questi comandi possono essere inseriti nel preambolo per modificare i parametri di tutti gli alberi oppure possono essere inseriti prima di un albero per modificare solo la sua struttura.

Nello specifico:

1. `\childsidegap{em}` imposta la distanza orizzontale tra bordi dei rami figli (1 em è il valore di *default*);
2. `\branchheight{in}` modifica la distanza verticale tra rami figli;
3. `\childattachsep{in}` imposta la distanza orizzontale tra i punti di derivazione dei rami figli (0.5 in è la distanza di *default* sia per questo valore che per il precedente);
4. `\trianglebalance{}` indica il bilanciamento del triangolo: con valore 50 (*default*) crea un triangolo isoscele, con valore 0 un triangolo rettangolo con l'ipotenusa a sinistra, con valore 100 un triangolo rettangolo con ipotenusa a destra.

3 Pacchetto xytrees

Nel caso di opere trasmesse in un'unica redazione, da una tradizione non contaminata o verticale, il pacchetto *synttree* è probabilmente l'opzione migliore per quanti vogliono disegnare uno *stemma codicum* con L^AT_EX.

Se però, come spesso accade, l'opera di cui si sta allestendo l'edizione critica ha avuto più di una fase redazionale, oppure se la tradizione presenta una certa contaminazione, occorre un pacchetto

che permetta anche la congiunzione orizzontale tra i nodi o l'unione di nodi di rami differenti.⁷

Il pacchetto `xytree` può essere d'aiuto quando si ha a che fare con *stemmata codicum* complessi e permette di risolvere molti problemi. Necessita, però, di un po' di studio, a ragione delle sue potenzialità.

3.1 Preliminari

Prima di disegnare un albero con il pacchetto `xytree` è opportuno immaginarne la struttura come fosse inserita in una sorta di piano cartesiano; i punti di questo piano sono individuati da coppie di numeri, separati da una virgola: quello che potremmo chiamare il "punto di origine", vale a dire il primo elemento posto nell'angolo in alto a sinistra corrisponde alla posizione 1,1, l'elemento alla sua sinistra dalla posizione 1,2, l'elemento posto immediatamente sotto il "punto di origine" 2,1 e così di seguito. Si veda la rappresentazione illustrata dalla figura 7.

1,1	1,2	1,3
2,1	2,2	2,3
3,1	3,2	3,3

FIGURA 7: Esempio di "matrice"

Ciò premesso, con questo pacchetto gli alberi si inseriscono con il comando `\xytree{ }` al cui interno occorre specificare l'intera struttura dello *stemma*. I singoli nodi vanno inseriti col comando `\xynode{ }` al cui interno si può inserire il testo da visualizzare in corrispondenza dei nodi. Come si diceva sopra, è opportuno immaginare la struttura dell'albero come se fosse inserita in un piano cartesiano o, se si preferisce, in una matrice: i nodi devono essere inseriti in una griglia, separati dal simboli `&` e `\\`, come gli elementi di una tabella. Si prenda l'esempio della figura 8.

Si può osservare che l'albero al momento non ha collegamenti tra i nodi ma, strutturalmente, è composto da tre *livelli* (potremmo definirli *righe*, a paragone di una tabella) e da tre *posizioni* (corrispondenti alle colonne di tabella); le singole *posizioni* sono separate dal simbolo `&`, i *livelli* da `\\`. Come per le tabelle, non è necessario tabulare il testo, ma è conveniente per poter controllare la struttura dell'albero.

7. Per i concetti di contaminazione, tradizione non meccanica e pluralità di redazioni si veda, almeno, PASQUALI (1934) e ITALIA e RABONI (2010).

```
\xytree{
    & \xynode{0} & \\
\xynode{A} & \xynode{B} & \xynode{C} \\
\xynode{D} & & \xynode{E}
}
```

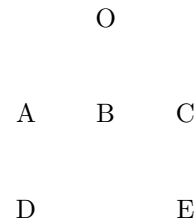


FIGURA 8: Albero senza collegamenti tra i nodi (`xytree`)

3.2 Collegamenti tra i nodi

3.2.1 Collegamenti semplici

L'albero mostrato nella figura 8 è sprovvisto di collegamenti tra i nodi. Per inserire i collegamenti basta inserire l'argomento opzionale al comando `\xynode`, indicando la posizione relativa del nodo del *livello* inferiore col quale si deve congiungere. Sembrava molto complesso, ma una volta capito il meccanismo è semplice. Ipotizzando di voler disegnare uno *stemma codicum* con tre testimoni derivanti da un unico capostipite, utilizzeremmo i comandi illustrati nell'esempio 9.

```
\xytree{
    &\xynode[-1,0,1]{0}& \\
\xynode{A}&\xynode{B} &\xynode{C}\\
}
```

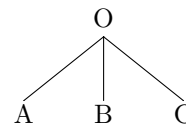


FIGURA 9: Esempio di *stemma* semplice (`xytree`)

Come si vede, abbiamo inserito l'opzione `[-1,0,1]` al nodo 0, volendo che, nel livello inferiore, i nodi inseriti nella posizione precedente a 0, corrispondente e successiva siano collegati con il nodo 0; in altre parole A occupa, nell'immaginaria matrice o piano cartesiano, la posizione -1 rispetto ad 0, B la medesima posizione di 0, C quella successiva, ovviamente al livello inferiore rispetto a 0.

Si prenda ora l'esempio illustrato nella figura 10 a fronte e si osservi che il comando `[0,2]` al nodo A produce un collegamento con il nodo D (stessa posizione relativa al livello inferiore) e con il nodo E (due posizioni dopo al livello inferiore); il comando `[-1]` al nodo B produce un collegamento con il nodo D (posizione relativa precedente al livello inferiore).

```
\xytree{
    & \xynode[-1,0,1]{O} &\
    \xynode[0,2]{A} & \xynode[-1]{B}%
    & \xynode[0]{C} \
    \xynode{D} &%
    & \xynode{E}
}
```

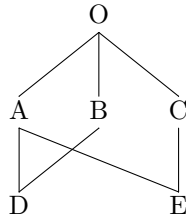


FIGURA 10: *Stemma* con alcuni collegamenti semplici tra nodi di due livelli successivi (`xytree`)

3.2.2 Collegamenti complessi

Il comando appena descritto, di facile utilizzo non appena capita la *ratio*, permette di realizzare dei collegamenti esclusivamente con nodi collocati al *livello* immediatamente inferiore: non è possibile ottenere collegamenti tra un nodo e uno posto a due o più *livelli* inferiori, come non è possibile collegare due nodi posti al medesimo *livello*.

Per ottenere collegamenti complessi si può utilizzare il comando `\xyconnect`. Questo comando ha cinque argomenti opzionali, tutti riguardanti il connettore:⁸

1. [tipologia del connettore];
2. [curvatura del connettore];
3. (punto di partenza, punto di arrivo);
4. {posizione del nodo di arrivo};
5. "etichetta".

Tipologia del connettore

Pur esistendo moltissime tipologie di collegamento,⁹ per i nostri fini ne elenchiamo solo 3:

1. [-] produce una linea continua (come i collegamenti visti negli esempi precedenti);
2. [->] produce una linea continua terminante con una freccia;
3. [...] produce una linea tratteggiata.

8. Il quinto argomento, relativo all'etichetta da inserire sul connettore, non verrà trattato.

9. Per le altre tipologie si veda UN (2006), p. 4; il manuale è visionabile digitando `texdoc xytree` da riga di comando.

Curvatura del connettore

Il secondo argomento ha tre possibili opzioni:

1. [^]:
2. [_]:
3. [_Npc]: con "N" valore numerico (in questo caso 1):

Punto di partenza, punto di arrivo

Le due indicazioni si riferiscono ai punti esatti di partenza e di arrivo del connettore rispetto al nodo: U (sopra), D (sotto), L (alla sinistra), R (alla destra) e le 4 posizioni intermedie (es. UL, sopra a sinistra...)

Alcuni esempi:

(U,D)

(R,L)

(D,D)

Posizione del nodo di arrivo

Indica la posizione del nodo da congiungere tramite il connettore; tale posizione può essere espressa sia come posizione assoluta, e allora deve essere indicata tra doppie virgolette alte, oppure come posizione relativa, e in questo caso l'indicazione deve essere posta tra parentesi graffe.

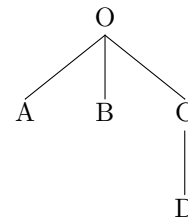


FIGURA 11: Esempio di *stemma* (`xytree`)

Si prenda lo *stemma* osservabile nella figura 11. Le seguenti sono le posizioni assolute e relative (rispetto a O) dei nodi:

A	"2,1"	{1,-1}
B	"2,2"	{1,0}
C	"2,3"	{1,1}
D	"3,3"	{2,1}

3.3 Esempi

Alle pagine che seguono si trova una galleria di esempi di stemmi realizzati col pacchetto `xytree`. Poiché spesso le famiglie di testimoni sono identificate da lettere greche, può essere utile inserire nel preambolo un comando sintetico per ottenere le lettere in greco (si ricorda che occorre inserire, tra le opzioni di `babel`, `polutonikogreek` per avere il greco politonico):¹⁰

10. Per un approfondimento su come scrivere in greco, si veda PANTIERI (2008), da cui si sono tratti questi suggerimenti.

```
\newcommand{\greco}[1]{%
\begin{otherlanguage*}%
{polutonikogreek}#1\end{otherlanguage*}}
```

Questo codice permette di avere a disposizione il comando `\greco{}`.

Alla tabella 1 si possono trovare le corrispondenze tra le lettere greche e italiane.

TABELLA 1: Corrispondenza lettere italiane e greche

a	b	c	d	e	f	g	h	i
α	β	ς	δ	ε	φ	γ	η	ι
j	k	l	m	n	o	p	q	r
ϑ	κ	λ	μ	ν	$\omicron$	π	χ	ρ
s(.)	t	u	v	w	x	y	z	
$\sigma(\cdot)$	τ	υ		ω	ξ	ψ	ζ	

4 Addenda

4.1 Dove collocare lo *stemma*?

Entrambi i pacchetti posizionano l'albero nella prima posizione utile stabilita da LATEX, vale a dire a un rientro dal margine se il comando `\synttree` o `\xytree` è separato dal testo precedente da una riga vuota, oppure sul margine se i comandi per disegnare l'albero non sono sperati dal testo da alcuna riga vuota. Ovviamente per centrarlo basta racchiudere l'albero nell'ambiente `center`, come si nota nell'esempio 12

```
\begin{center}
\synttree
[A
[B ]
[D [D] [E [F]] [G] ]
]
\end{center}
```

FIGURA 12: Comandi per centrare lo *stemma* (non elegante)

Sia questa soluzione, sia il comportamento di *default*, però, potrebbero non essere molto convenienti, né del tutto eleganti: in entrambi i casi l'albero si trova ad essere un "oggetto in testo" e quindi non è prevista una didascalia né riferimenti espliciti da altre parti del documento.¹¹ Lo *stemma codicum* necessita spesso di una didascalia e può essere molto utile potervi fare riferimento direttamente.

Per ottenere queste funzioni, è possibile inserire l'albero in un ambiente `figure` (per comodità l'esempio si riferisce al pacchetto `synttree`, ma analogo discorso vale anche per `xytree`) come si può vedere nell'esempio illustrato dalla figura 13.

11. PANTIERI e GORDINI (2012), p. 86.

```
\begin{figure}[htb]
\centering
\synttree
[A
[B ]
[C [D] [E [F]] [G] ]
]
\caption{\emph{Stemma} nell'ambiente%
\texttt{figure}}
\label{esempio1}
\end{figure}
```

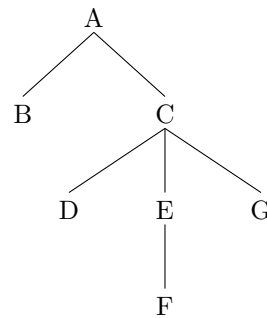


FIGURA 13: *Stemma* nell'ambiente `figure`

Per richiamare lo *stemma* si dovranno utilizzare i consueti comandi; nel caso specifico: `\vref{esempio1}` (`\vref` è un comando del pacchetto `varioref`).

Se si vuole modificare la dicitura “Figura” presente nella didascalia degli alberi, si deve intervenire sulle stringhe generate da `babel`, inserendo nel preambolo questi comandi:

```
\addto\captionsitalian{\renewcommand%
{\figurename}{Stemma}}
\addto\captionsitalian{\renewcommand%
{\listfigurename}{Elenco degli stemmi}}
```

Questo tipo di intervento modifica le didascalie di tutte le figure, pertanto è possibile servirsene solo nel caso in cui, in tutto il documento, non si debbano inserire delle figure diverse dagli stemmi.

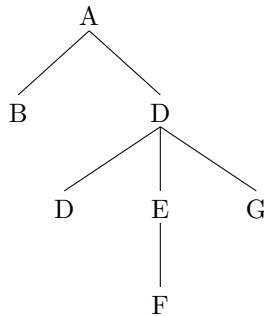
In alternativa si può creare un nuovo ambiente utilizzando il pacchetto `newfloat`.

Inserendo nel preambolo questo codice:

```
\usepackage{newfloat}
\DeclareFloatingEnvironment[
fileext=los,
listname={Elenco degli stemmi},
name=Stemma,
placement=htb,
within=section,
]{stemma}
```

si hanno a disposizione sia il comando `\listofstemmas` che collocato dopo `\tableofcontents` stampa l'*Elenco degli stemmi*, sia l'ambiente `stemma` da utilizzare come descritto nello *stemma* 1 nella pagina successiva.

```
\begin{stemma}
\synttree
[A
[B ]
[D [D] [E [F]] [G] ]
]
\caption{Esempio dell'ambiente%
\texttt{stemma}}
\label{stemma1}
\end{stemma}
```



STEMMA 1: Esempio dell'ambiente `stemma`

Si rimanda alla documentazione del pacchetto per le necessarie modifiche da apportare al nuovo ambiente.¹²

4.2 Stemmi di grande dimensione

Può accadere che lo *stemma* di un testo sia particolarmente complesso e abbia numerose ramificazioni: il risultato sarà uno *stemma* più largo della pagina. Per ovviare a questo inconveniente basta utilizzare il pacchetto `rotating`, in questo modo:

```
\begin{sidewaysfigure}
\centering
\synttree
[A
[B ]
[D [D] [E [F]] [G] ]
]
\caption{\emph{Stemma} ruotato di 90°}
\label{Esempio_ruotato}
\end{sidewaysfigure}
```

L'ambiente `sidewaysfigure` stampa lo *stemma* ruotandolo di 90°: si veda la documentazione del pacchetto per le opzioni relative, come sempre, richiamabile con il comando `texdoc rotating`.

5 Conclusioni

Questo contributo ha presentato due pacchetti che permettono la realizzazione di stemmi, uno di facile utilizzo (`synttree`) e uno molto più complesso ma davvero versatile (`xytree`); oltre a questo si è cercato di fornire alcuni suggerimenti per gestire

12. Visionabile digitando `texdoc newfloat` da riga di comando.
sia la collocazione degli stemmi sia gli eventuali alberi di grande dimensione.

In appendice all'articolo si trova una galleria di esempi nella quale si possono trovare tre stemmi ipotetici di opere con una pluralità di redazioni (figure da 14 a 16 nella pagina seguente) e la riproduzione di tre stemmi di opere letterarie, la *Commedia* (figura 17 a pagina 121) e il *De Monarchia* (figura 19 a pagina 122) di Dante e le *Catilinarie* di Cicerone (figura 18 a pagina 121).

Il presente contributo non si pone l'obiettivo di fornire una panoramica di tutti i pacchetti che permettono la creazione di stemmi, quali `qtree` o `TikZ`. Nel presente numero di ArT_EXnica si può trovare un contributo a firma Leal-Pignalberi che descrive `TikZ` (pubblicato alla pagina 123 e seguenti, LEAL e PIGNALBERI (2012a)).

Riferimenti bibliografici

- CICERO, M. T. (1905). *Orationes*. Clarendoniano.
- DAIN, A. (1975). *Les manuscrits*. Les Belles-Lettres.
- ITALIA, P. e RABONI, G. (2010). *Che cos'è la filologia d'autore*. Carocci.
- LEAL, J. e PIGNALBERI, G. (2012a). «Composizione di uno *Stemma codicum* con `TikZ`». ArT_EXnica, 14.
- (2012b). *Edizioni critiche. Guida alla composizione con il proprio computer*. Edizioni CompoMat.
- MANIACI, M. (2002). *Archeologia del manoscritto. Metodi, problemi, bibliografia recente*. Viella.
- PANTIERI, L. (2008). *L'arte di scrivere in greco con L^AT_EX*. URL http://www.lorenzopantieri.net/LaTeX_files/Greco.pdf.
- PANTIERI, L. e GORDINI, T. (2012). *L'arte di scrivere con L^AT_EX*. URL http://www.lorenzopantieri.net/LaTeX_files/ArteLaTeX.pdf.
- PASQUALI, G. (1934). *Storia della tradizione e critica del testo*. Le Monnier.
- PIGNALBERI, G. e BINI, E. (2008). «L^AT_EX e grammatiche (la faccia triste dell'informatica)». ArT_EXnica, 6, pp. 76–85.
- QUAGLIONI, D. (2011). «Un nuovo testimone per l'edizione della *Monarchia* di Dante: il Ms. Add. 6891 della British Library». *Laboratoire italien*, 11, pp. 231–280. URL <http://laboratoireitalien.revues.org/595>.
- SANGUINETI, F. (a cura di) (2001). *Dantis Alagherii Comedia*. Edizioni del Galluzzo.
- STUSSI, A. (1994). *Introduzione agli studi di filologia italiana*. il Mulino.

TIMPANARO, S. (1963). *La genesi del metodo di Lachmann*. Le Monnier.

UN, K. (2006). *xytree.sty*. A $\text{\LaTeX}$ package for drawing syntactic trees.

▷ Matteo Fadini
Dipartimento di Studi letterari,
linguistici e filologici
Università degli Studi di Trento
matteofadini at gmail dot com

```
\xytree{
    & \xynode[-1,0,1]{0} \xyconnect[.] (D,D){0,2} & & \xynode[0]{0\ap{1}} \\\
    \xynode{A} & \xynode{B} & \xynode{C} & \xynode{D}
}
```

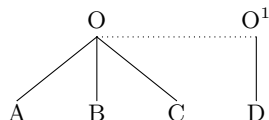


FIGURA 14: Esempio di *stemma* di un'opera con due redazioni.

```
\xytree{
    & \xynode[-1,0,1]{0} \xyconnect[.] (D,D){0,2} & & \xynode[0]{0\ap{1}} \\\
    \xynode[0]{A} & \xynode[0]{B} & \xynode{C} & \xynode[0]{D} \%
    \xyconnect[.] (D,U){1,-2} \\\
    \xynode{E} & \xynode{F} \xyconnect[.] (U,D){-1,1} & & \xynode{G} \\\
}
```

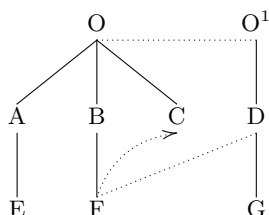


FIGURA 15: Esempio di *stemma* di un'opera con due redazioni e con testimoni contaminati

```
\xytree{
    & \xyconnect[.] (D,D){0,2} \xynode[-1,0,1]{0} & \%
    & \xynode[0]{0\ap{1}} & & \\\
    \xyconnect[.] (D,L){1,2} \xynode[0]{A} & \xynode{B} & \xynode[0]{C} & \xynode[0]{D} \%
    \xyconnect[.] (D,D){0,2} & & \xynode[-1,1]{0\ap{2}} & \\\
    \xynode{E} & & \xynode{F} & \xynode{G} & \xynode{H} & & \xynode{I} \\\
}
```

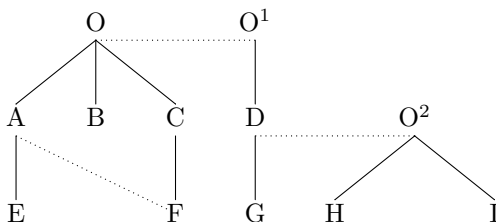


FIGURA 16: Esempio di *stemma* di un'opera con tre redazioni e una modesto grado di contaminazione.

```
\xytree{
&&&\xynode[5,-3]{\greco{w}} & & & & \\\
\xynode[3]{\greco{a}}\xyconnect[-](D,U){3,0} && & & & & \xynode{\greco{b}}}%
\xyconnect[-](D,U){4,0}\\\
&& &\xynode[2]{x}\xyconnect[-](D,U){2,-2} && & & \\\
&& & &\xynode[-2,1]{z} && & \\\
\xynode[0]{y}& \xynode[0,1]{a}& &\xynode[0,1]{b} && &\xynode[0,1]{c} && \\\
\xynode{LauSC}&\xynode{Mart}&\xynode{Triv} &\xynode{Ham} &\xynode{Ash} & %
&\xynode{Mad} & \xynode{Rb} & \xynode{Urb} \\\
}
```

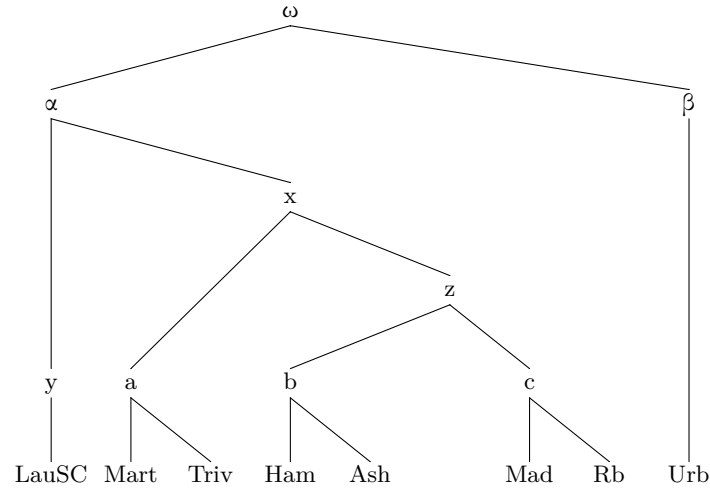


FIGURA 17: *Stemma codicum della Commedia* tratto da SANGUINETI (2001)

```
\xytree{
&&&&\xynode[-3,2,5]{\greco{w}}&&&&& \\\
&&\xynode[-1,1,0,-2]{\greco{a}} \xyconnect[.](D,U){2,3}%
&&&&\xynode[-1,-2,0,1]{\greco{b}}\xyconnect[.](D,U){1,2}&&&%
\xynode[0,1]{\greco{g}} \xyconnect[.](D,U){1,-1}& \\\
\xynode{C}&\xynode{A}&\xynode{V}&\xynode{h}&&\xynode{o}&\xyconnect[.](D,U){1,0}&\xynode{t}%
&\xynode{u}&\xynode{x}&\xynode{l}&\xynode{b}&\xynode{s} \\\
&&&&\xynode{a}&&&&& \\\}
```

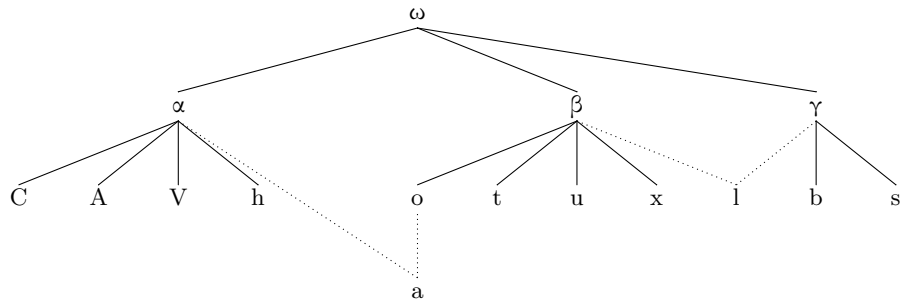


FIGURA 18: *Stemma codicum della Catilinarie* tratto da Cicerone (1905)

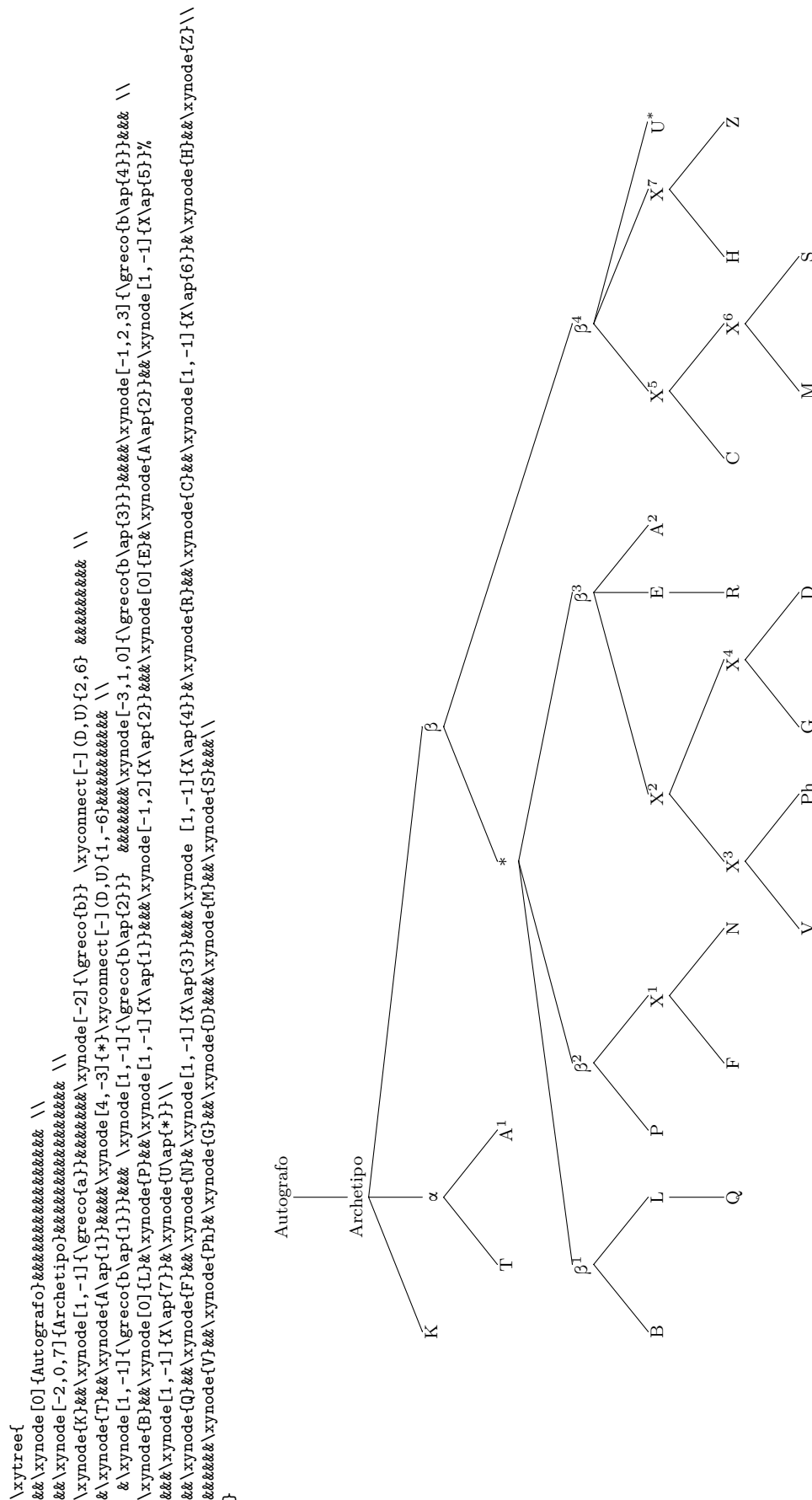


FIGURA 19: *Stemma codicum* del *De Monarchia* tratto da QUAGLIONI (2011)

Composizione di uno *Stemma codicum* con TikZ

Jerónimo Leal, Gianluca Pignalberi

Sommario

Sebbene L^AT_EX abbia ottimi pacchetti dedicati alla composizione di uno *stemma codicum* più o meno complesso, TikZ fa ottenere risultati almeno altrettanto buoni senza bisogno di introdurre maggior complessità notazionale.

Abstract

While L^AT_EX has very good packages to typeset a more or less complex *stemma codicum*, TikZ allows to reach at least results as good without requiring harder notational complexity.

Premessa

Gli autori di questo articolo, pensato poche settimane dopo la pubblicazione del loro libro (LEAL e PIGNALBERI, 2012), lo hanno completato solo dopo la presentazione dell'articolo di Matteo Fadini presente su questo stesso numero di *ArXiv*. Quell'articolo ha convinto gli autori a esplorare ulteriormente le capacità di TikZ e a tentare di realizzare gli stessi esempi complessi.

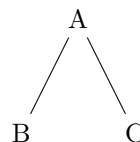
I risultati confutano i timori espressi da Fadini in una conversazione privata (*TikZ potrebbe non essere in grado di comporre gli stessi schemi complessi di xytree*).

Questa premessa è anche un esplicito ringraziamento allo stesso Fadini per aver ispirato e discusso con gli autori le parti dell'articolo non previste (e neanche pensate) in origine.

1 Introduzione

L'elaborazione di uno *stemma codicum* richiede strumenti adeguati per restituire chiaramente le dipendenze dei manoscritti. L'articolo di Matteo Fadini (pubblicato alla pagina 113 e seguenti, FADINI (2012)) descrive *synttree* (PIGNALBERI e BINI, 2008; VAN ZUIJLEN, 2009) e *xytree* (UN, 2006). Mentre il primo è adatto a costruire solo schemi molto semplici quali sono gli alberi,¹ il secondo è in grado di costruire schemi più complessi come i grafi in cui un nodo può derivare da più genitori. La notazione imposta da *xytree* non ha la stessa semplicità di quella di *synttree* e richiede all'utente di avere già uno schema mentale esatto da riprodurre: mentre

1. Un albero è una struttura dati formata da nodi — elementi con un contenuto — e da archi — collegamenti tra i nodi. Ogni nodo di un albero può appartenere a uno e uno solo dei nodi al livello superiore, cioè avere un solo genitore. L'unica eccezione è la radice: occupa il livello supremo.



```
\begin{tikzpicture}
\node{A}
  child{node{B}}
  child{node{C}}
;
\end{tikzpicture}
```

FIGURA 1: Albero con due sole foglie (B e C) figlie della radice (A).

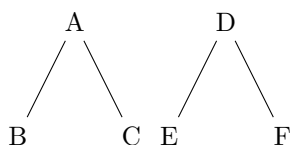
synttree chiede solo di elencare ogni nodo dell'albero seguito dai suoi figli (*organizzazione logica*) per generare da sé l'albero, *xytree* chiede l'esatta posizione spaziale di ogni nodo in un'ipotetica griglia (o tabella) e l'elenco di tutti i collegamenti, o archi (*organizzazione fisica*). Dunque l'utente è l'unico responsabile dell'aspetto finale del grafo.

TikZ (TANTAU, 2010) offre un'interfaccia intermedia: dobbiamo indicare l'elenco dei nodi seguiti dai figli, cosa che genera direttamente l'albero, dopodiché possiamo specificare una serie di collegamenti supplementari applicabili solo ai nodi precedentemente etichettati. Nel corso di questo breve articolo chiariremo tutti i dettagli notazionali ricorrendo ad alcuni esempi concreti tratti da una tesi del dottorando Fabrizio Tiddia in preparazione alla Pontificia Università della Santa Croce e consistente nell'edizione critica della *Passio Sancti Alexandri*.

Il paragrafo 2 introduce la produzione di uno o più alberi con TikZ. Il paragrafo 3 spiega come segnalare le contaminazioni tra i manoscritti mediante l'aggiunta di archi arbitrari. Infine il paragrafo 4 espone come comporre con TikZ alcuni degli *stemma codicum* realizzati da Matteo Fadini nel suo articolo FADINI (2012).

2 Costruzione di alberi semplici con TikZ

Un albero è una struttura comunemente usata per rappresentare delle strutture gerarchiche: è costituito da *nodi* e *archi*, dove i nodi sono dei simboli contenenti una didascalia, cioè un'informazione, mentre gli archi sono dei segmenti che uniscono due nodi di due livelli adiacenti. Un albero contie-



```
\begin{tikzpicture}
\node{A}
  child{node{B}}
  child{node{C}}
;
\node at (2,0) {D}
  child{node{E}}
  child{node{F}}
;
\end{tikzpicture}
```

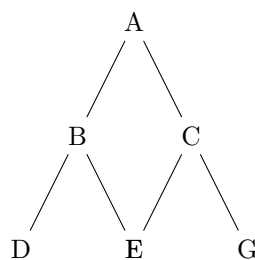
FIGURA 2: Due alberi nello stesso ambiente tikzpicture.

ne sempre una *radice*, un nodo senza “genitore”, da cui possono dipartire diversi rami, quindi con eventuali *figli*. I figli che non hanno a loro volta dei figli sono definiti *foglie*.

La sintassi di TikZ ricalca la nomenclatura esposta, come vediamo nell'esempio mostrato nella figura 1: dentro l'ambiente `tikzpicture` apriamo (radichiamo) l'albero col comando `\node` a cui diamo una didascalia (da non confondere con l'etichetta, come vedremo più avanti), in questo caso “A”. Quindi elenchiamo tutti i suoi figli dal primo (quello più a sinistra) all'ultimo (quello più a destra): generiamo ognuno dei figli con l'attributo `child`. Questo prende tra parentesi graffe l'attributo `node`, la didascalia (anch'essa tra parentesi graffe), e l'eventuale lista dei figli del nodo corrente. La descrizione dell'albero termina con un ‘;’.

Niente ci vieta di disegnare due o più alberi nello stesso ambiente `tikzpicture`, come vediamo nella figura 2. Dobbiamo solo avere l'accortezza di specificare la posizione della radice degli alberi successivi al primo per non averli tutti sovrapposti a esso.

Naturalmente non siamo vincolati a produrre alberi con una profondità predeterminata dato che possiamo aggiungere tutti livelli che servono. La figura 3 mostra lo stesso albero della figura 1 in cui però le vecchie foglie ora hanno dei figli. Noterete che i rami relativi alle foglie “E” e “F”, provenienti da due *parenti* (due o più nodi allo stesso livello) adiacenti, sembrano puntare alla stessa foglia. In realtà le foglie sono due sovrapposte perché TikZ, a differenza di `syntree`, non tiene conto automaticamente degli ingombri, che devono essere corretti manualmente specificando la distanza tra parenti. Vediamo come applicare la correzione nella figura 4. Per ognuno dei livelli da “ritoccare” alteriamo uno degli attributi di `style`: `sibling distance`. Se avessimo specificato, ad esempio, solo l'attributo del primo livello (`level 1`), tutti i successivi livelli avrebbero ereditato il suo valore, quindi anche i



```
\begin{tikzpicture}
\node{A}
  child{node{B}
    child{node{D}}
    child{node{E}}}
  child{node{C}
    child{node{F}}
    child{node{G}}}
;
\end{tikzpicture}
```

FIGURA 3: Albero con una radice, due figli (nodi intermedi) e quattro foglie (nodi terminali). Le due foglie denominate “E” e “F” sono sovrapposte.

nodi del secondo livello avrebbero avuto una distanza tra essi di 40 mm anziché 20. Ogni livello è separato dal successivo da una virgola e tutte le descrizioni sono passate come parametro opzionale all'ambiente `tikzpicture`.

Dopo aver visto alcuni casi didattici vediamo un caso concreto tratto dalla tesi citata. La figura 5 mostra uno *stemma codicum* il cui disegno non pone problemi: non ci sono due parenti adiacenti che abbiano figli, quindi non ci sono possibilità di sovrapposizioni e dunque non c'è la necessità di inserire correzioni di sorta.

Ricordiamo che quello che abbiamo finora chiamato “radice” sarà più correttamente definito *archetipo*. Abbiamo già discusso la sintassi di un albero in TikZ, quindi non ci soffermeremo in ulteriori descrizioni. L'unico appunto che possiamo fare riguarda la stesura del codice: usiamo l'indentazione solo per comodità umana. Il compilatore non necessita di spazi particolari ma solo del codice conforme alle regole del linguaggio.

Passare a uno *stemma codicum* con un maggior numero di livelli non cambia la difficoltà (sebbene servirà maggiore attenzione nello scrivere tutte le parentesi correttamente). Lo stemma mostrato nella figura 6 è notevolmente più grande di quelli visti finora, didattici o no. Se è vero che la maggior grandezza è motivo di maggior scrittura di codice, è anche vero che il disegno non presenta alcuna difficoltà, visto che non ci sono sovrapposizioni da correggere né altre caratteristiche che discuteremo nel paragrafo 3: siamo ancora nel caso di alberi.

Quello che piuttosto possiamo notare è che nel codice alcuni nodi hanno una lettera tra parentesi tonde dopo l'attributo `node`. Questa è un'etichet-

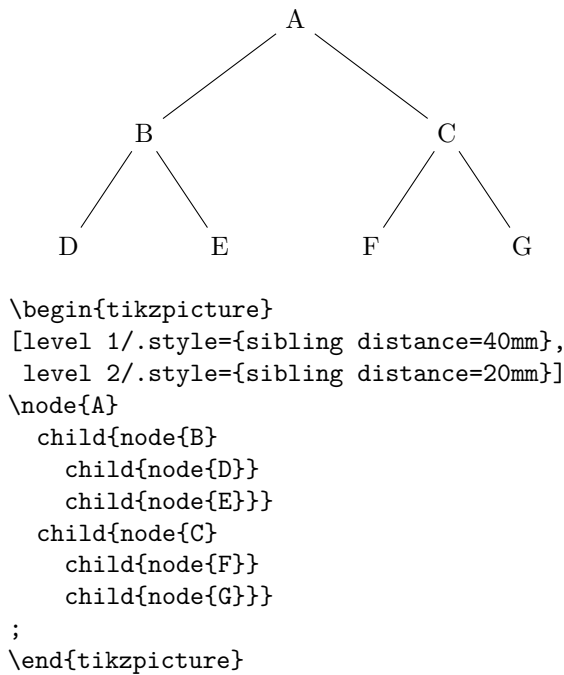


FIGURA 4: Albero con una radice, due figli (nodi intermedi) e quattro foglie (nodi terminali) dopo la correzione manuale degli ingombri.

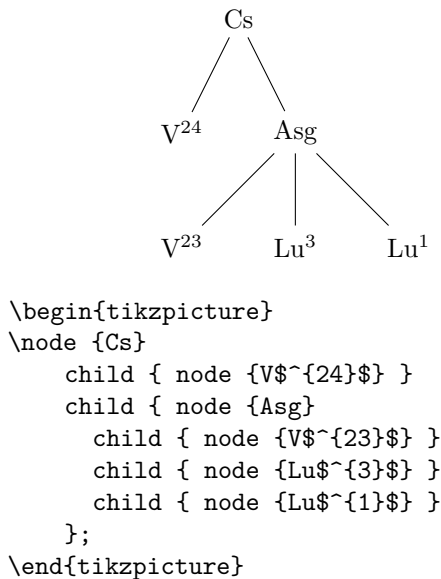


FIGURA 5: Uno *stemma codicum* concreto.

ta che serve per futuri riferimenti ai nodi quando aggiungeremo degli archi, cioè dei percorsi non previsti nell'albero standard. Ovviamente, un albero che non rispetti più le regole costruttive e costitutive di tale struttura, come avere archi che uniscano nodi arbitrari, non sarà più un albero ma un grafo. Proprio di grafi ci occuperemo nel paragrafo 3 in cui introdurremo le *contaminazioni*.

Abbiamo già visto come possiamo variare la distanza tra i nodi dello stesso livello. TikZ ci permette di variare anche la distanza tra i livelli

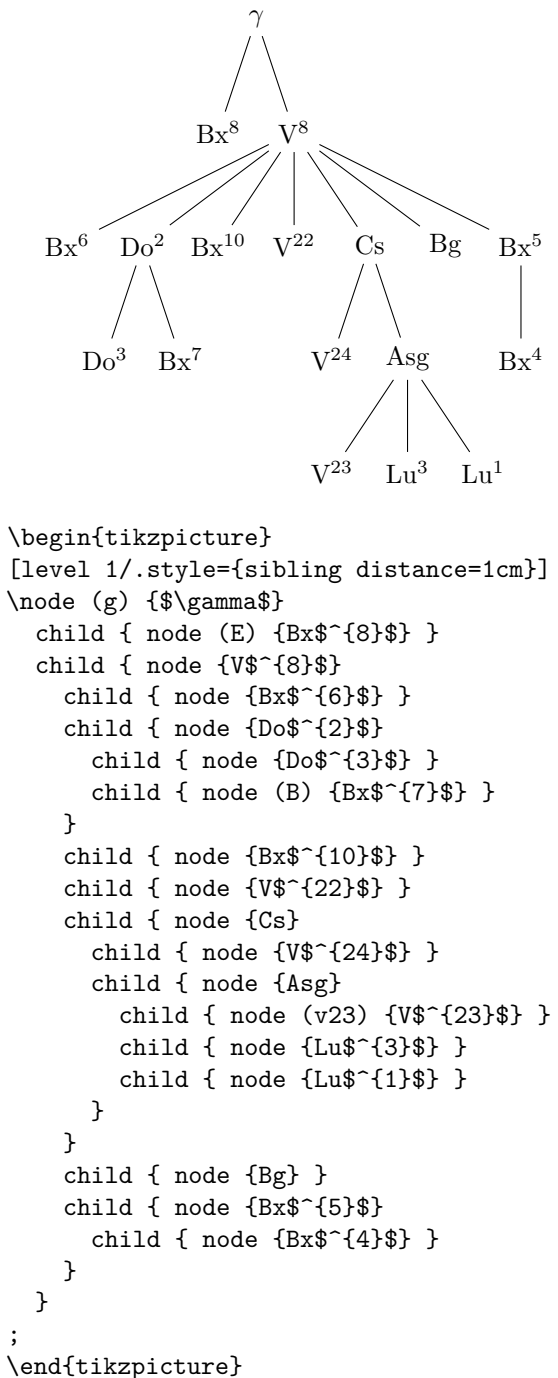


FIGURA 6: Uno *stemma codicum* più profondo (cinque livelli).

grazie all'attributo `level distance`. La sintassi è uguale a quella vista per `sibling distance`: basta specificare il valore dell'attributo come opzione all'apertura dell'ambiente `tikzpicture`. Anche in questo caso l'attributo è ereditario e viene propagato agli altri livelli, a meno che non specifichiamo un valore diverso per ogni livello.

Se vogliamo imporre che tra un livello e l'altro la distanza sia di 3 cm dobbiamo scrivere

```

\begin{tikzpicture}[level 1/.style={level distance=3cm}]

```

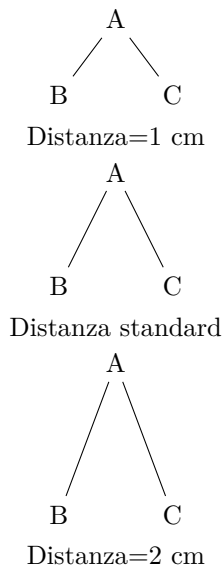


FIGURA 7: Alberi con i livelli a distanze differenti.

La figura 7 mostra lo stesso albero con diverse distanze tra i livelli.

3 Segnalazione delle contaminazioni

Abbiamo detto che TikZ ci permette di etichettare i nodi per disegnare degli archi non previsti dalla costruzione standard dell'albero. Per comodità etichetteremo solo i nodi di interesse, cosa già vista nella figura 6, e aggiungeremo il codice relativo al disegno degli archi *dopo* il ';' che chiude la costruzione dell'albero:

```
\draw[dashed,-] (B) -- (E);
\draw[dashed,-] (B) -- (g);
\draw[dashed,-] (E) -- (v23);
\draw[dashed,-] (g) -- (v23);
```

Il comando `\draw` serve a disegnare qualcosa; nel caso specifico disegna un segmento rettilineo (`--`) tra due nodi (le lettere tra parentesi). Come argomento opzionale di `\draw` specifichiamo lo stile del segmento (qui *dashed*, cioè tratteggiato) e il tipo di terminazione. In questo caso non c'è alcuna terminazione perché il segno `-` indica un segmento senza terminazioni. avremmo potuto omettere l'informazione essendo questo il caso di default. Vediamo il relativo risultato nella figura 8.

Quanto visto finora relativamente alla connessione dei nodi può essere applicato al caso di due alberi separati senza alcuna difficoltà: la figura 9 mostra proprio un caso del genere.

Questo espediente ci permetterà di collegare lo stemma concreto a un archetipo α posto alla sua sinistra. Aggiungeremo il comando

```
\node (a) at (-5,0) { $\alpha$ };
;
```

prima dell'albero iniziale o dopo, o anche dopo gli archi delle contaminazioni.

Sempre a causa del fatto che TikZ non tiene conto degli ingombri, le linee tratteggiate delle contaminazioni attraversano dei nodi rendendo confusa la lettura dello *stemma codicum*. Possiamo modificare la posizione dei nodi o la curvatura degli archi in diversi modi: quello più immediato è spostare leggermente la posizione del contenuto dei nodi, magari introducendo dello spazio vuoto prima o dopo il contenuto. Spazi forzati, `box`, `\phantom` o altro sono tutti espedienti rientranti in questa categoria e utilizzabili con successo.

Non ci concentriamo però su questi metodi ma sugli strumenti propri di TikZ, un potentissimo pacchetto che fornisce gli strumenti necessari per curvare gli archi. Invece di usare un arco rettilineo (indicato con il simbolo `--` tra due punti o due etichette nel comando `\draw`), useremo un arco curvilineo indicato con il simbolo `..` e uno o due *punti di controllo*. I punti di controllo indicano le coordinate verso cui puntano le tangenti al punto iniziale e al punto finale del segmento da disegnare. Se non specifichiamo il secondo punto, TikZ lo assumerà uguale al primo. Ci troviamo quindi di fronte a una curva di Bézier quadratica (un punto di controllo) o cubica (due punti) (BECCARI, 2012; WIKIPEDIA, 2012). Separiamo i due punti di controllo con `and`. Ad esempio,

```
\draw[dashed,-] (g) .. controls (.1,-2.5)
and (.3,-2) .. (v23);
```

Risulta chiaro che i punti di controllo servono solo a definire una curvatura; in generale la curva non passerà per essi.

Nel caso qui esposto abbiamo scelto i punti di controllo nei pressi dei nodi attraversati. Ne troviamo facilmente le coordinate tenendo conto che la radice a cui non abbiamo specificato `at` viene posta in (0,0) (origine). Le coordinate seguono la convenzione del piano cartesiano: la prima è positiva per i punti a destra dell'origine e negativa per quelli alla sua sinistra; la seconda è positiva per i punti sopra l'origine e negativa per quelli sotto di essa.

Ammettiamo che l'utenza diversa dagli scienziati (fisici, matematici, informatici, chimici), dagli ingegneri e da chi ha un buon retroterra matematico e geometrico può rimanere atterrito dalla spiegazione precedente. Può essere un incubo trovare i punti di curvatura per chi non "mastica" correntemente i numeri e le coordinate cartesiane. TikZ soccorre questi malcapitati mettendo a disposizione una griglia numerata come aiuto alla determinazione delle coordinate. Purtroppo dobbiamo disegnarla e numerarla da noi, ma è un'operazione alla portata di tutti.

In un ambiente `tikzpicture` disegneremo per prima cosa la griglia, grigia e con le linee fine per maggior contrasto col disegno principale e un po'

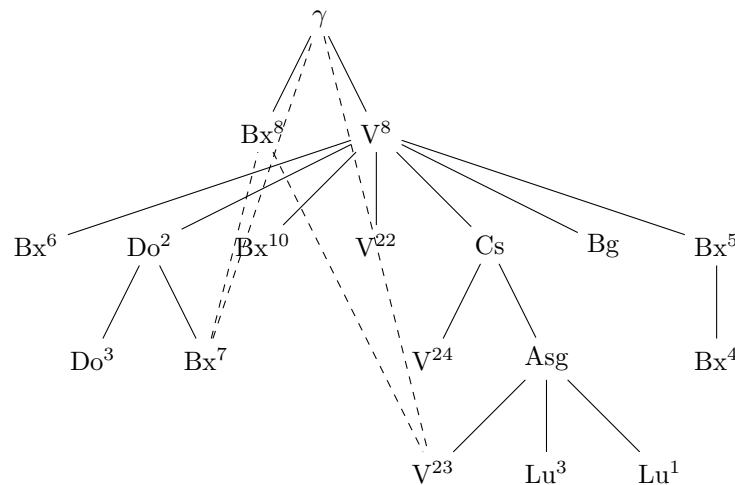
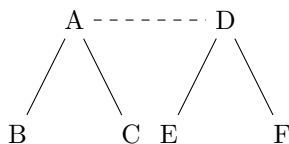


FIGURA 8: *Stemma codicum* con alcune contaminazioni segnate con archi rettilinei. Capita che questi archi si sovrappongano ad alcuni nodi.



```
\begin{tikzpicture}
\node(A){A}
  child{node{B}}
  child{node{C}}
;
\node at (2,0) (D){D}
  child{node{E}}
  child{node{F}}
;
\draw [dashed] (A) -- (D);
\end{tikzpicture}
```

FIGURA 9: Due alberi separati con le radici connesse.

più grande dello stemma così da contenerlo completamente. Il comando `\draw` accetta il punto iniziale e finale divisi dall'attributo `grid`. Opzionalmente possiamo dare il *passo* della griglia (la distanza tra due linee adiacenti), il colore e lo spessore delle linee. Questo comporterà alcune prove iniziali ma renderà più facile il lavoro; cancelleremo la griglia appena il grafo sarà definitivo. Dopo le prime volte che avremo usato questo “bastone d'appoggio” arriveremo addirittura a farne a meno per sempre. Nel nostro caso la griglia adatta è

```
\draw[step=1cm,gray,thin] (-5,-7)
  grid (6,1);
```

Possiamo rendere la griglia millimetrata anziché centimetrata aggiungendo

```
\draw[step=1mm,gray,very thin] (-5,-7)
  grid (6,1);
```

al precedente comando. Teniamo presente che possiamo anche sostituire i due attributi `gray` e `very thin` con `help lines` ottenendo un effetto simile.

A questa griglia dovremo aggiungere i numeri. Potremmo scriverli uno per uno ma, nel caso la griglia abbia molte linee, la cosa diventerebbe impraticabile. Sfruttiamo quindi il comando per i cicli di TikZ: `\foreach`. Con un ciclo scriviamo i numeri sull'asse verticale e con un altro ciclo scriviamo i numeri sull'asse orizzontale:

```
\foreach \y in {-7,...,1}
  \node at (-5.5, \y)
    {\makebox[1pc][r]{\y}}; % numeri
    % imbandierati a destra
\foreach \x in {-5,...,6}
  \node at (\x, 1.5) {\x};
```

Dopo il comando `\foreach` specifichiamo un contatore (nel nostro caso sono due: `\x` e `\y`) che assumerà di volta in volta uno dei valori possibili, elencati tra parentesi graffe. Siccome i valori possono essere molti e sarebbe impraticabile elencarli tutti, possiamo specificare solo i punti di inizio e fine separati dalle virgole e da `...`. Questo definisce un *intervallo* di valori. In questo caso semplice i valori dell'intervallo differiscono di 1: il primo contatore assumerà valori -7, -6, -5 e così via fino a 1, mentre il secondo assumerà i valori -5, -4, -3 e così via fino a 6. L'unica istruzione eseguita da ogni ciclo è il disegno di un nodo composto da una scritta: il valore del contatore tra parentesi graffe. La posizione del nodo è data dal valore del contatore e da una coordinata fissa, spostata di poco rispetto agli assi di riferimento. La figura 10 mostra il risultato della griglia con dentro lo *stemma codicum* da modificare: ora è facilissimo capire dove prendere i punti di controllo per curvare gli archi tratteggiati.

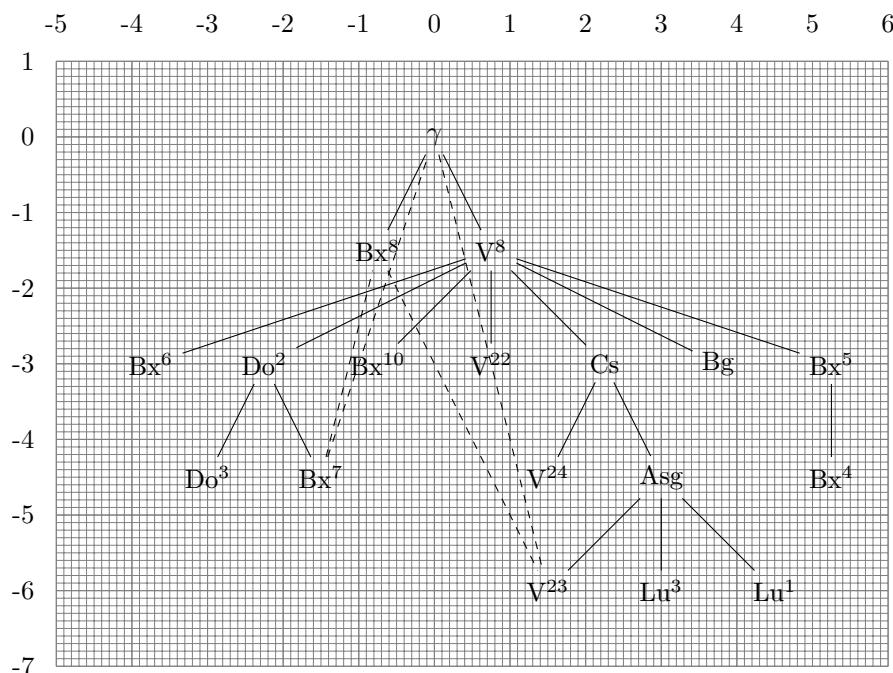


FIGURA 10: La griglia sullo sfondo dello *stemma codicum* rende più agevole la determinazione dei punti di controllo per curvare gli archi sovrapposti ai nodi.

Quello che faremo, ad esempio, è sostituire i seguenti comandi:

```
\draw[dashed,-] (B) ..
    controls (-1.2,-2.75) .. (E);
\draw[dashed,-] (B) ..
    controls (-1.25,-1) .. (g);
\draw[dashed,-] (E) -- (v23);
\draw[dashed,-] (g) ..
    controls (.1,-2.5) .. (v23);
\draw (a) -- (g);
```

a quelli usati per disegnare lo stemma della figura 8. Il risultato è visibile nella figura 11. L'ultimo `\draw` serve a unire “ α ” e “ γ ”.

4 Contro xytree

Siamo arrivati alla “sfida” *TikZ* contro *xytree*, sfida limitata a tre degli esempi mostrati da Matteo Fadini in FADINI (2012), riprodotti nella figura 12 e affiancati all’equivalente versione composta con *TikZ*. Vediamo di seguito i sorgenti di entrambe le versioni, ricordando che le versioni *xytree* sono riprodotte in accordo con il loro autore e indentate diversamente in virtù dei diversi spazi.

Stemma 1

Il codice `xytree` è il seguente:

```
\xytree[ & \xynode[-1,0,1]{O}
\xyconnect[.](D,D){O,2} & &
\xynode[0]{O\ap{1}} \\\
\xynode{A} & \xynode{B} &
\xynode{C} & \xynode{D}
}
```

mentre il codice `TikZ` è questo:

```
\begin{tikzpicture}
\node(0){0}
  child{node {A}}
  child{node {B}}
  child{node {C}}
;
\node at (3,0) (01){0$^1$}
  child{node {D}}
;
\draw[dotted] (0) -- (01);
\end{tikzpicture}
```

Stemma 2

Il codice xytree è il seguente:

```
\xymatrix{ & \\ \xymatrix[-1,0,1]{0} \\ \xymatrixconnect[.](D,D){0,2} & & \\ \xymatrix[0]{0\ap{1}} \\ \xymatrix[0]{A} & \xymatrix[0]{B} & \\ \xymatrix{C} & \xymatrix[0]{D} \\ \xymatrixconnect[.](D,U){1,-2} \\ \xymatrix{E} & \xymatrix{F} \\ \xymatrixconnect[.>]{\sim}(U,D){-1,1} & & \\ \xymatrix{G} \\ }
```

mentre il codice `TikZ` è questo:

```
\node(0){0}
  child{node {A}
    child{node {E}}}}
  child{node {B}
    child{node {F}{F}}}}
```


mentre il codice TikZ è questo:

```
\begin{tikzpicture}
\node(0){0}
  child{node (A){A}
    child{node {E}}}
  child{node {B}}
  child{node {C}
    child{node (F){F}}}
;
\node at (3,0) (01){0$^1$}
  child{node (D){D}
    child{node {G}}}
;
\node at (5,-1.5) (02){0$^2$}
  child{node {H}}
  child{node {I}}
;
\draw[dotted] (0) -- (01);
\draw[dotted] (A) -- (F);
\draw[dotted] (D) -- (02);
\end{tikzpicture}
```

Dai codici appena visti è evidente quanto detto nell'introduzione: disegnare uno *stemma codicum* con *xytree* è veramente come comporre una tabella con un nodo per cella, connessioni a parte. Invece, disegnarlo con TikZ è più simile a comporre un albero con *synttree* poiché TikZ mantiene la sintassi di un albero (con in più la complessità e le possibilità del suo linguaggio) a cui vanno poi aggiunte le eventuali connessioni supplementari.

Quale dei due metodi sia migliore, però, dipende da ogni autore e dal suo stile di ragionamento, di organizzazione spaziale e di preferenza visiva. Ognuno può vedere i risultati a confronto (codici e disegni) e decidere per sé quale metodo usare caso per caso o definitivamente.

5 Conclusioni

Gli utenti L^AT_EX hanno a disposizione diversi modi per comporre uno *stemma codicum*. Il più semplice di essi, il pacchetto *synttree*, permette la generazione di un albero a partire da un codice di descrizione dell'albero stesso (organizzazione logica). Tranne i casi più semplici, uno *stemma codicum* concreto sarà ben più complesso di un albero: dovrà mostrare tutte le contaminazioni sia tra diversi archetipi che tra documenti successivi. Ciò trasforma un albero in un grafo. Purtroppo *synttree* non è in grado di gestire questa complessità perché serve a comporre degli alberi mentre *xytree* è più adatto in casi del genere. Tuttavia la sua sintassi è più simile a quella di una tabella che non a quella di un grafo (organizzazione fisica). A tale proposito TikZ offre una soluzione più adatta poiché la sintassi dei suoi comandi per generare alberi e grafi è più simile a quella di *synttree*, con l'ulteriore possibilità di personalizzare l'aspetto finale del disegno.

L'articolo ha mostrato come comporre degli *stemma codicum* con TikZ, ha esposto come sia possibile connettere più alberi e come segnalare le contaminazioni etichettando i nodi dell'albero e connettendoli *ad hoc*. Ha inoltre spiegato come agguistare le distanze e come deformare gli archi per evitare sovrapposizioni. Infine ha descritto come replicare alcuni risultati permessi dal pacchetto *xytree*. Tutto ciò ha trovato e trova tutt'ora riscontro in casi concreti quali gli *stemma codicum* della tesi di dottorato di Fabrizio Tiddia.

Riferimenti bibliografici

- BECCARI, C. (a cura di) (2012). *Introduzione all'arte della composizione tipografica con L^AT_EX*. Versione A4-0.98t.
- FADINI, M. (2012). «*Stemma codicum* con L^AT_EX». *ArsTeXnica*, (14).
- LEAL, J. e PIGNALBERI, G. (2012). *Edizioni Critiche. Guida alla composizione con il proprio computer*. T_EXNOLOGIE. CompoMat, Rieti, Italia.
- PIGNALBERI, G. e BINI, E. (2008). «L^AT_EX e grammatiche (la faccia triste dell'informatica)». *ArsTeXnica*, (6), pp. 76–85. URL <http://www.guitex.org/home/images/ArsTeXnica/AT006/LATEX%20e%20grammatiche.pdf>.
- TANTAU, T. (2010). *The TikZ and PGF Packages*. URL <http://www.ctan.org>. Lo leggete dando `texdoc tikz` al prompt del terminale o di DOS.
- UN, K. (2006). *xytree.sty — A L^AT_EX package for drawing syntactic trees*. URL <http://www.ctan.org>. Lo leggete dando `texdoc xytree` al prompt del terminale o di DOS.
- VAN ZUIJLEN, M. (2009). *The synttree package for typesetting syntactic trees*. URL <http://www.ctan.org>. Lo leggete dando `texdoc synttree` al prompt del terminale o di DOS.
- WIKIPEDIA (2012). «Curva di Bézier». URL http://it.wikipedia.org/wiki/Curva_di_Bézier.

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

Alcuni stralci sul Computer Bodoni

Claudio Vincoletto

Sommario

Il tentativo di riprodurre, mediante una metamorfosi del Computer Modern di Knuth, alcuni alfabeti realizzati da Giambattista Bodoni che rappresentano l'apice del suo stile.

Abstract

The Computer Modern font source may be conveniently modified in order to obtain a new version of a Bodoni type nearer to the original one.

1 Tra esotismo ed archeologia

Per comprendere appieno le scelte messe in opera in questo studio è necessario dapprima delineare un quadro storico. In questo senso, tra gli indizi di gusto neoclassico ravvisabili nell'opera di Giambattista Bodoni (1740–1813), si possono sicuramente annoverare le serie di alfabeti definiti “esotici”.

A partire dal saggio poliglotta, in occasione delle nozze di Ferdinando di Borbone nel 1769, in cui mette a frutto lo studio delle lingue orientali apprese durante l'apprendistato romano, e quello per il battesimo del principe Ludovico del 1774, Bodoni affianca all'esperienza di stampatore quella di incisore di caratteri, mettendosi in evidenza appunto con i caratteri esotici. Successivamente il campionario viene ampliato a dismisura, fino ad esplodere nella celebre *Oratio Dominica* (in 155 lingue) o nel postumo *Manuale Tipografico* del 1818 (con ben 123 lingue esotiche, più due “tedeschi” ossia gotici).

Ancora oggi sfogliare questi volumi può rappresentare un'esperienza stupefacente, se si considerano la varietà delle lingue illustrate e i corrispondenti caratteri approntati per la pubblicazione. E sempre perfettamente in linea con la moda in voga all'epoca, in cui la tendenza all'esotismo affiora nella letteratura come nella musica, oltre che nell'architettura e nella scultura; un arco di tempo che va dal *Vathek* (1786) al *Manuscrit trouvé à Saragosse* (1805), o altrimenti dal *Die Entführung aus dem Serail* (1782) a *L'italiana in Algeri* (1813).

Tra i caratteri forgiati in queste raccolte, che più o meno si allontanano dai romani e dai corsivi usati comunemente oggi, figurano alcuni alfabeti che appartengono a lingue morte: etrusco, fenicio, punico. Per poco egli non reinventò pure i geroglifici, sulle orme di Athanasius Kircher. Anche questi alfabeti antichi vengono considerati esotici da Bodoni, nel pieno rispetto dei canoni neoclassici. Per gli intellettuali d'Olttralpe i paesi del Mediterraneo

(Italia, Grecia, Spagna) rappresentavano non solo luoghi lontani, ma anche testimonianze di un passato antiquario pronto ad emergere dalle rovine abbandonate. L'antico è essenzialmente esotico, ma proiettato nel tempo invece che nello spazio.

Gli scavi di Pompei ed Ercolano, intrapresi allora, avevano incuriosito il mondo intero lanciando una nuova moda e al tempo stesso i fondamenti di una nuova scienza. La pubblicazione del *Recueil d'antiquités égyptiennes, étrusques, grecques et romaines* di Caylus del 1752, delle *Antichità di Ercolano*, a cura dell'Accademia Ercolanense fondata da Carlo III (1757–92) e dai *Vasi, candelabri, cippi, sarcofagi, tripodi...* di Piranesi (1778), consolidarono il diffondersi del gusto neoclassico almeno fino all'inizio dell'Ottocento.

Tale rinascita e fermento culturale influirono su Bodoni in modo obliquo, almeno per ciò che concerne il disegno dei caratteri. Il suo ritorno al passato punta idealmente alla nascita della tipografia, piuttosto che alle autentiche forme delle scritture lapidarie greco-romane, che del resto erano già state a lungo studiate nel corso del Rinascimento. Per delineare i suoi alfabeti romani, Bodoni non si discosta molto dalla sua epoca, che segna il definitivo passaggio dalle forme transizionali a quelle moderne. Il suo riferimento estetico però saranno sempre i primi stampatori, Jenson, Manuzio, Garamond, così come si evince chiaramente dai suoi scritti.

Il marcato contrasto, che rende molto scuro lo specchio della pagina, e la foggia austera dei glifi, sono gli aspetti più riconoscibili dello stile bodoniano. Ricordano le consuetudini calligrafiche tardo medievali, età in cui nasce la stampa, quasi un riconsiderare il *textura* di Gutenberg, ma perfezionato in caratteri romani moderni. Esperimento eccentrico forse, eppure non così assurdo, se si concepisce come mitizzazione nostalgica di un mondo ideale ormai perduto, ovvero come recupero archeologico che attinge dal periodo “classico” della tipografia. «Con caratteri nuovi, scriviamo pagine antiche», parrebbe suggerire Bodoni, parafrasando André Chénier.

2 Bodoni calligrafo

Petrus Scriverius, nel suo *Laure-crans voor Laurens Coster* del 1628, volle immortalare lo stampatore olandese con un carattere tipografico fra le dita, nello specifico la lettera A, quale leggendario inventore della stampa.

Possiamo dedurre che il punto di vista di Bodoni fosse diametralmente opposto. In molti ritratti infatti viene raffigurato mentre impugna una penna d’oca, come un perfetto gentiluomo dedicatosi allo studio delle belle lettere. Del resto, come tutti i celebri tipografi del passato, Bodoni fu principalmente un letterato.

Il fatto che Bodoni fosse dedito alla calligrafia, e che da essa prendesse spunto per l’incisione dei suoi punzoni, non è un’ipotesi priva di fondamento.

Nel suo *The alphabet* del 1918, Frederic Goudy afferma a proposito di Bodoni:

Fece i tratti sottili più fini e le aste più spesse di quanto fosse mai stato fatto per i precedenti caratteri romani, e incise le sue lettere con una precisione ed una regolarità mai eguagliate prima. Considerava i suoi disegni come fossero stati eseguiti con una penna a punta larga, ma con un pennino così ampio che mai nessuno aveva tentato di utilizzare. La qualità chiara e regolare dei suoi caratteri diede alle sue stampe una perfezione ed una brillantezza che sono talvolta abbacinanti.

Daniel B. Updike, in *Printing Types* (1922), ripubblica una celebre lettera del 1787, inviata da Benjamin Franklin allo stampatore italiano, che ha risollevato la questione circa la presunta attitudine calligrafica di Bodoni. A Updike interessa solo il valore documentario della lettera, per dimostrare quanto fosse celebre Bodoni ai suoi tempi. In essa, il famoso inventore si lamenta, pur ossequioso, del disegno di alcuni caratteri apparsi nella stampa della *Lettre à M. le Marquis de Cubières*, che a suo parere sarebbero incoerenti rispetto al naturale movimento che è proprio della scrittura manuale.

L’argomento discusso nella lettera permette a Gerrit Noordzij, nel suo *Letterletter* (2000), di portare il discorso su un altro livello, ossia di affermare che ancora al tempo di Bodoni era proprio la calligrafia a dettare lo standard per il disegno dei caratteri. Naturalmente questo criterio dovrebbe essere tenuto in considerazione per ogni riproduzione di tipi bodoniani, anche digitale.

Bodoni avrebbe comunque seguito una tradizione consolidata, così come i suoi contemporanei, ma reinterpretandola e rinnovandola in modo originale e definitivo. Il suo successo come stampatore di corte, avulso dalle logiche commerciali del tempo, nel passaggio tecnologico dalla stampa periodica – tipica del Settecento – all’industria del libro – trionfante nell’Ottocento –, e prigioniero di un ideale di bellezza rivolto principalmente alla costruzione della pagina e ai frontespizi, ha messo in secondo piano la sua qualità di creatore di caratteri, che non di rado è stata poco apprezzata.

Il fraintendimento nasce soprattutto nel Novecento, quando ogni fonderia di caratteri propone

una propria versione di caratteri bodoniani, tradendo irrimediabilmente il disegno originale. Grazie a piatte che non conservano la benché minima modulazione, proporzioni esageratamente geometriche, un contrasto amplificato all’eccesso tanto da ostacolare la lettura (ancor più evidente nei font digitali privi di scalatura ottica, con il ben noto effetto *dazzle*, per cui l’alternanza di tratti spessi e sottili ostacola la leggibilità dei caratteri ridotti a piccole dimensioni).

Eppure Bodoni mirava a tutt’altro, tenendo sempre in mente che la tipografia non è altro che una declinazione della scrittura. Nell’Introduzione al *Manuale Tipografico* del 1818, dichiara esplicitamente:

Come in ogni altra cosa, così pur anche nella scrittura la moda regna e dà leggi, talor con ragione, e talor senza. Ove però buona ragion non appaja, e la moda non tiranna lasci arbitrio, il buon gusto si attiene a una semplicità non rozza, quale si mostrerebbe delineando con tratti per tutto egualmente grossi le lettere, ma ben avvisata e gentile, quale scorgesi nel bel contrasto per dir così di chiari e scuri, che vien naturale a ogni scritto di ben tagliata penna e ben tenuta mano.

3 Perché METAFONT?

Anche in questo caso si potrebbe trattare di un ritrovamento archeologico, poiché oggi, tra coloro che disegnano caratteri di professione, METAFONT è sottovalutato se non, nella maggior parte dei casi, misconosciuto.

Qui viene considerato come lo strumento più opportuno per affrontare i tipi bodoniani.

Propriamente METAFONT va considerato come un linguaggio, fondato su un lessico di termini univoci ed una grammatica piuttosto rigida, capace di descrivere con precisione l’aspetto dei vari glyphi. Questi sono raccolti in un programma, il font, che istruisce il compilatore ad elaborare i caratteri come immagini bitmap. Ideato nei primi anni Ottanta da Donald Knuth, in congiunzione con TEX, presenta ancora oggi delle peculiarità che lo rendono potentissimo.

Per lavorare con METAFONT è necessario immaginare uno spazio bidimensionale su cui fissare dei punti, definiti come coordinate cartesiane, attraverso i quali vengono tracciati dei tratti che li relazionano, esattamente come accade quando si scrive su un pezzo di carta. Una volta impostata la forma ed il comportamento dello strumento scrittorio, idealmente una penna, è necessario indicarne lo spostamento e la rotazione per svilupparne il tratto, ovvero dare luogo alla scrittura. In questo senso METAFONT è pensato su principi calligrafici,

anche se in effetti può riprodurre qualsiasi disegno grafico e simboli di vario genere.

Delineare le forme generate per mezzo di parametri costituisce un ulteriore elemento distintivo di METAFONT. Ogni caratteristica, lo spessore del tratto, l'andamento delle curve, il contorno delle grazie, viene esaminata singolarmente, dotandola di un valore preciso che corrisponde ad unità misurabili. Il variare questi parametri influisce pesantemente sul risultato finale, per cui a partire da una struttura di base, lo scheletro di ogni lettera, si possono ottenere famiglie complete di caratteri, composte da alfabeti romani in diversi corpi ottici, corsivi, in grassetto, senza grazie e così via.

In questo senso si rivela la natura di METAFONT e il suo essere "meta", al di sopra, oltre il font vero e proprio. Si chiarisce perciò come, intervenendo su una serie di parametri, sia possibile avviare un processo di mutazione a partire da un carattere esistente, il Computer Modern, per trasformarlo in qualcosa di completamente diverso, nel nostro caso il Computer Bodoni.

4 Sviluppare il Computer Bodoni

Per Computer Bodoni si intende una famiglia di caratteri digitali che mira a riprodurre alcuni alfabeti originali di Giambattista Bodoni con l'ausilio di METAFONT.

Il disegno del Computer Modern romano deriva dal *Modern 8a* della Monotype, una versione "americana" del carattere Modern, che a sua volta consiste in un'interpretazione riduttiva e contemporanea dei caratteri di Didot, Bodoni e Walbaum. La struttura di base non si presenta quindi così differente dai tipi di Bodoni, ma molti dettagli rendono talvolta irriconoscibile questo rapporto.

Addirittura l'uso di METAFONT permette, grazie all'elevato livello di astrazione, un'analisi accurata di tali alfabeti e in questo senso può essere considerato anche un valido strumento didattico, se non un veicolo che favorisce nuovi approcci interpretativi all'intera opera di Bodoni.

4.1 Le fonti

Ha poco senso parlare di autenticità quando ci si riferisce ad una riproduzione di caratteri in digitale. Si parte sempre da una fonte analogica, ad esempio un libro stampato nell'Ottocento, che viene ridotta in digitale per velocizzarne l'elaborazione, quindi riversata in analogico, un'odierna stampa casalinga. Una grande quantità di informazioni viene perduta; altri aspetti ritenuti cruciali sono invece conservati. Il problema sta nel determinare cosa verrà conservato.

In una stampa a torchio influiscono diversi fattori sulla percezione della pagina, come la qualità della carta, l'umidità della carta nel momento dell'impressione, la composizione chimica dell'inchiostro utilizzato, la pressione del torchio, l'usura dei

caratteri tipografici e via dicendo. Di volta in volta questi elementi si combinano in modo casuale, tanto che risulta difficile catturare la forma esatta di ogni glifo, che quindi richiede un'interpretazione.

Sicuramente la qualità delle fonti può influire sull'intero processo. Questo è il motivo per cui il Computer Bodoni deriva direttamente da reali alfabeti utilizzati da Bodoni nelle sue stampe e non da una sintesi artificiale del suo stile, come accade in molte versioni di font in commercio. Il modello a cui si fa riferimento è riconducibile a pochi corpi, presenti nel *Manuale Tipografico* del 1818, che presentano una certa unitarietà.

La selezione ha anch'essa una sua tradizione, di cui si vuole fare brevemente cenno. Nel 1922 Giovanni Mardersteig ottenne dalle autorità italiane, con non poche difficoltà, il diritto di utilizzare una scelta di matrici originali di Bodoni. Ecco la ragione per cui la prestigiosa stamperia, fondata a Montagnola di Lugano, fu chiamata Officina Bodoni. Per molti anni questi furono i soli caratteri usati da Mardersteig, decretandone in parte il successo.

Il primo approccio al Computer Bodoni è avvenuto attraverso alcune scansioni in scala di grigi da fac-simile provenienti dall'Officina Bodoni e dal *Manuale Tipografico* del 1818. Tali scansioni devono raggiungere una risoluzione di almeno 2400 dpi, soprattutto per i corpi più piccoli. Successivamente è stato possibile accedere alla ricca collezione bodoniana, conservata presso la Biblioteca Civica di Torino, dove sono disponibili diversi manoscritti e stampe originali, tra cui un esemplare del *Manuale Tipografico* che si distingue per un'ottima inchiostatura, piuttosto nitida.

I corpi considerati sono il Messina (10 pt), il Cuneo (12 pt), il Loreto (14 pt), soprattutto il Catania (16 pt, figura 1) ed infine il Casale (20 pt). Tutte le misure vanno considerate in punti Didot.

4.2 L'elaborazione delle immagini

Le scansioni sono state elaborate con GIMP in vari modi, al fine di farne risaltare alcuni particolari.

L'immagine va dapprima ridimensionata, rifilandoli i margini ed eliminando le parti superflue, per diminuirne il peso complessivo (strumento Ritaglia). Soprattutto è opportuno allinearla orizzontalmente, in modo che la linea di base su cui poggiano i caratteri risulti essere parallela al riquadro della finestra (strumento Ruota).

In genere, le immagini che presentano una minore luminosità ed un contrasto ridotto sono preferibili, poiché conservano un maggior numero di informazioni. GIMP offre l'opportunità di variare questi parametri in modo dettagliato, migliorando di conseguenza le operazioni dei filtri che gestiscono il rilevamento dei contorni. Fra tutti, l'opzione Affilatura si dimostra molto efficace nel definire il profilo delle figure che si presenta quindi più netto; i valori scelti dipendono ovviamente dalla qualità



FIGURA 1: Specimen del Computer Bodoni, Catania.

della scansione originale e vanno reimpostati ogni volta.

È possibile ancora aggiungere all'immagine un livello separato su cui applicare il rendering automatico di una quadrettatura o griglia. Si ripropone quindi una tecnica simile a quella impiegata diffusamente negli affreschi rinascimentali, ma che risale addirittura all'Antico Egitto, per mantenere le proporzioni di una figura a partire da uno schizzo su carta. Dettando opportune istruzioni, anche METAFONT può visualizzare un reticolato molto simile, per cui ogni singola lettera dell'alfabeto viene a stagliarsi su uno sfondo che ricorda la carta millimetrata. Il confronto fra l'immagine elaborata dalla scansione e quanto prodotto con METAFONT risulta dunque enormemente facilitato.

4.3 La trasformazione del Computer Modern

Per capire come rimaneggiare i file sorgente del Computer Modern è essenziale scoprirne il funzionamento. Il font si articola in una serie di file con estensione `.mf`, ai quali possono essere attribuite tre funzioni diverse. Innanzitutto vi sono i file da cui si avvia la compilazione, che sono riconducibili ai vari corpi che si desiderano ricavare. Ad esempio, il file `cmr10.mf` contiene tutti i parametri che consentono di ottenere il Computer Modern romano a 10 pt (maiuscole, minuscole, punteggiatura, simboli). Vi è poi un livello intermedio in cui figurano i disegni delle singole lettere, ossia la loro struttura, che valgono come riferimento per diversi corpi e stili di carattere. Il file `roman1.mf` (contrazione di *roman lowercase*) viene richiamato ogni qual volta si desiderino dei caratteri minuscoli romani corrispondenti a svariati corpi, ma anche per quelli senza grazie o in grassetto, come richiesto dal file che li invoca. Infine esiste un unico file,

il `cmbase.mf`, il quale raccoglie tutti gli elementi comuni che si ripetono di lettera in lettera, ad esempio la foggia delle grazie, oppure le indicazioni per la conversione in pixel, cui è destinato il carattere finale.

Per evitare conflitti con il Computer Modern tutti i file necessari sono stati copiati e rinominati, per identificarli come Computer Bodoni. Successivamente si è intervenuto nel mutare i parametri di ogni corpo, in particolare del Catania, di cui era disponibile nelle immagini un maggiore numero di glifi. Una volta raccolte mediante GIMP le misure dei parametri (strumento Misurino) e ricopiate in un foglio di calcolo, queste vengono convertite in valori che risultino comprensibili per METAFONT.

Il foglio di calcolo utilizzato per questa operazione è Calc della suite LibreOffice. In ogni colonna vengono distribuiti i valori in pixel, così come rilevati con GIMP. La dimensione dell'occhio della lettera, che deve corrispondere al corpo in punti Didot, viene assunta come fattore di conversione, per cui tutti i parametri vengono ricalcolati in base a questo rapporto. I nuovi valori vanno a sostituire quelli originali del Computer Modern, dando vita di fatto ad un nuovo font.

Per quel che concerne la struttura delle figure, si interviene cambiando la posizione dei punti cardine che caratterizzano ogni lettera, in modo empirico, tenendo sempre presente la quadrettatura di cui si è già discusso. Il valore `u#`, parametro di METAFONT che influenza come unità fondamentale la larghezza di ogni glifo, viene impiegato fattivamente per la costruzione della griglia. L'adozione di `u#` come punto di riferimento grafico viene già contemplata dal Computer Modern, ma solo per la componente verticale. Nel Computer Bodoni sono state integrate le linee orizzontali, per collimare con la griglia riprodotta con GIMP.

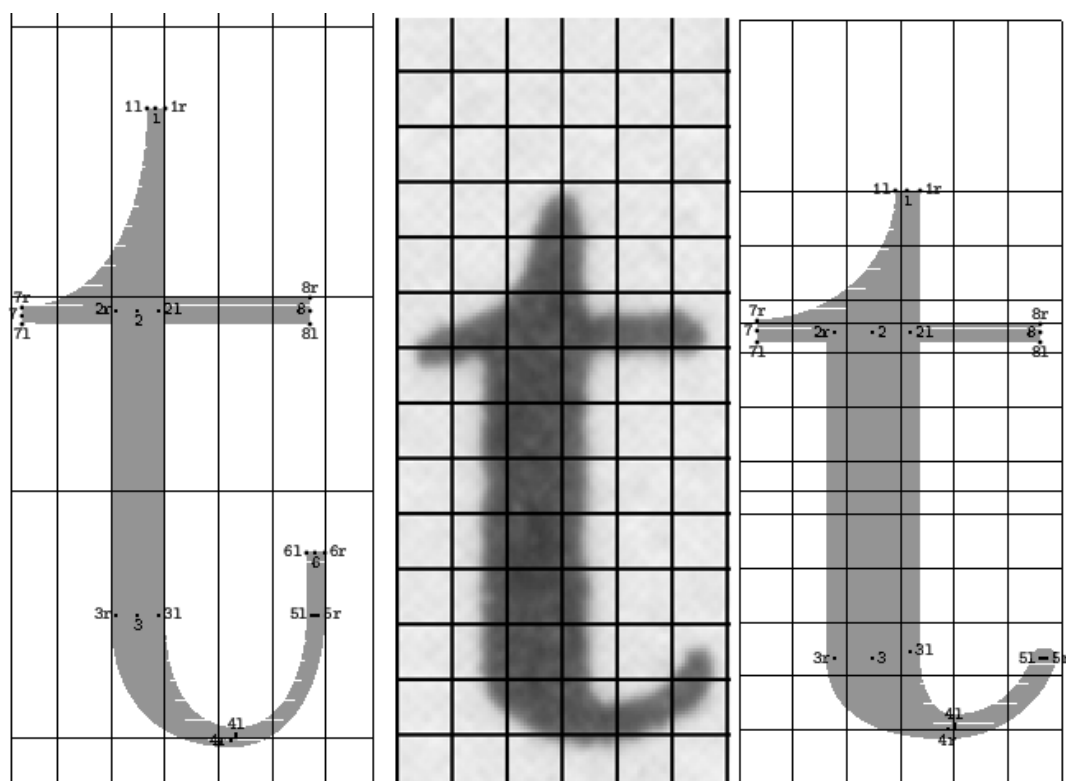


FIGURA 2: Confronto tra Computer Modern, Catania e Computer Bodoni.

Alcuni glifi non necessitano di alcuna correzione, vista la parentela con il Computer Modern, altri invece vanno ritoccati, anche in modo sostanziale. Allo stesso tempo si può lavorare sul `bdnbase.mf` (così è il nuovo nome), per intervenire globalmente su alcune grazie o, ad esempio, per visualizzare la griglia.

Le righe del codice originale sono conservate nel file modificato e commentate, ossia invisibili in fase di compilazione, per il confronto e la valutazione delle correzioni apportate.

Verrà illustrato qui di seguito un esempio, che si spera esplicativo.

Si tratta della lettera “t”, nel confronto fra il Computer Modern a 17 punti anglosassoni e il Computer Bodoni a 16 punti Didot. Nel mezzo, l’immagine della lettera così come si presenta nel Catania (figura 2).

Si può subito notare come la variazione del tratto appaia più robusta nei tipi di Bodoni. Questo effetto si può ottenere a partire dai parametri fondamentali, presenti nel `Cbodoni-rm16.mf`. Eccone un breve estratto (le righe commentate appartengono al Computer Modern), che indica la misura delle aste rette, dei filetti, quindi delle curve.

```
%hair#:=11/36pt#; % lowercase hairline breadth
hair#:=14.4/36dd#; % lowercase hairline breadth

%stem#:=33/36pt#; % lowercase stem breadth
stem#:=53.5/36dd#; % lowercase stem breadth

%curve#:=40/36pt#; % lowercase curve breadth
curve#:=61.7/36dd#; % lowercase curve breadth
```

Per cambiare la posizione di alcuni nodi, si interviene invece sul file `Cbodoni-roman1.mf`, che sostituisce il `roman1.mf` del Computer Modern.

Ad esempio nel tratto di testa, che si presenta meno slanciato nel Catania, viene spostata l’ordinata relativa al punto 1. Ovvero si riduce il valore del rapporto tra l’altezza dell’ascendente, definita da `asc_height`, e l’altezza della x, `x_height`.

(Per comodità vengono spezzate le righe troppo lunghe, mediante il simbolo “\” ed un rientro nella riga successiva).

```
% min(asc_height#,if hefty:9/7 else:10/7 \
fi\\ x_height#),0);
min(asc_height#,if hefty:9/7 else:9.3/7 \
fi\\ x_height#),0);
```

Anche il gancio terminale del Computer Modern risulta essere troppo prominente rispetto all’originale bodoniano. Qui è sufficiente commentare le righe di codice che definiscono il tratto tra il punto 5’ (identico al punto 5) ed il punto 6, di fatto rendendolo invisibile.

```
% pickup crisp.nib; pos6(hair,0); pos5’(hair,0);
% x6=x5=x5’; top y6=max(vround .75bar_height,top \
y5); y5=y5’;
% filldraw stroke z5’e--z6e; % terminal
```

Per mezzo di svariate correzioni, analoghe a quelle illustrate, è possibile quindi ricreare un nuovo glifo, che si discosta abbastanza dal disegno di Knuth per avvicinarsi maggiormente a quello di Bodoni. Nel lavoro svolto si è mantenuta il più

tinis patinas ingentes extis mullorum refer-
tas et cerebellis foenicopterum et perdicum
ovis et cerebellis turdorum et capitibus psit-
tacorum et fasianorum et pavonum. 7 Bar-
bas sane mullorum tantas iubebat exhibe-
ri, ut pro nasturtiis, apiasteris et faselaribus
et feno Graeco exhiberet plenis fabatariis et
discis. Quod praecipue stupendum est.

XXI. 1 Canes iecineribus anserum pavit.
Habuit leones et leopardos exarmatos in de-
liciis, quos edoctos per mansuetarios subi-
to ad secundam et tertiam mensam iubebat
accumbere ignorantibus cunctis, quod exar-
mati essent, ad pavorem ridiculum excitan-
dum. 2 Misit et uvas Apamenas in praesepia
equis suis et psittacis atque fasianis leones
pavit et alia animalia. 3 Exhibuit et sumi-
na apruna per dies decem tricena cottidie
cum suis vulvis, pisum cum aureis, lentem
cum cerauniis, fabam cum electris, orizam
cum albis exhibens. 4 Albas praeterea in vi-

FIGURA 3: Esempio del Computer Bodoni. Aelius Lampridius, *Vita Antonini Heliogabali*, dall'*Historia Augusta*.

possibile l'impalcatura di base del Computer Modern, privilegiando una visione d'insieme, talvolta trascurando alcuni dettagli tipici del Catania, che però non si riscontrano negli altri corpi presi in esame.

5 Conclusioni

Sperimentare la trasformazione dei file del Computer Modern può costituire un ottimo apprendistato per chi volesse approfondire l'uso di METAFONT, oltre naturalmente alla consultazione del manuale di Knuth.

Allo stato presente, il lavoro sul Computer Bodoni si presenta ancora parziale e lacunoso, ma promettente. Superata la fase preliminare, sicuramente passibile di miglioramenti, verranno valutati eventuali contributi in vista di un progetto collaborativo più esteso.

In ogni caso, con pochi esempi, si è ribadito quanto METAFONT sia in grado di produrre facilmente font di alta qualità. Non può quindi essere relegato all'oblio solo per la complessità della sua sintassi o perché incapace di produrre al volo caratteri in formato .ttf o .otf, odierni standard del mercato. Esistono soluzioni antiche e moderne per ovviare a quest'ultimo limite, dall'impiego di METAPOST al più recente MFLua. Quanto alla difficoltà di apprendere il funzionamento, si tratta davvero di poca cosa se paragonata allo sforzo di capire cosa sia la scrittura e renderla al meglio: uno degli scopi imprescindibili dell'arte tipografica.

Rivisitare Bodoni, alle soglie del bicentenario della morte, non può che agevolare questo proposito.

Riferimenti bibliografici

- DE LAMA, G. (1816). *Vita del cavaliere Giambattista Bodoni: tipografo italiano, e catalogo cronologico delle sue edizioni*. Dalla Stamperia ducale. URL <http://archive.org/details/vitadelcavalier01dela>.
- FUSSEL, D. (2010). *Bodoni: Manuale Tipografico*. Taschen.
- GOUDY, F. (1963). *The alphabet, and Elements of lettering*. Dover Publications. URL <http://archive.org/details/cu31924020595934>.
- HARALAMBOUS, Y. e HORNE, P. (2007). *Fonts & Encodings*. O'Reilly Media, Incorporated.
- KNUTH, D. (1986). *The METAFONTbook*. Addison-Wesley.
- MARDERSTEIG, G. (1988). *Scritti di Giovanni Mardersteig sulla storia dei caratteri e della tipografia*. Il Polifilo.
- NOORDZIJ, G. (2000). *Letterletter*. Hartley & Marks.
- PAZ, M. (1940). *Gusto neoclassico*. G.C. Sansoni.
- UPDIKE, D. (1922). *Printing Types, Their History, Forms, and Use: A Study in Survivals*. v. 2. Harvard University Press. URL <http://archive.org/details/printingtypethe01updi>.

▷ Claudio Vincoletto
Torino
claudio dot vincoletto at
gmail dot com

Questa rivista è stata stampata
presso Braille gamma S.r.l., Cittaducale (RI)
su carta Revive silk 115 g
Copertina su carta Revive natural 300 g



ArTeXnica – 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 ArTeXnica, per essere sottoposto alla valutazione di recensori entro e non oltre il 14 Marzo 2013. È necessario che gli autori utilizzino la classe di documento ufficiale della rivista; l'autore troverà raccomandazioni e istruzioni più dettagliate all'interno del file d'esempio (.tex).

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

Di volta in volta verrà fissato, e reso pubblico sulla pagina <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@sssup.it.

ArsT_EXnica

Numero 14, Ottobre 2012

- 5 Editoriale
Gianluca Pignalberi
- 7 Beyond BibT_EX
Jean-Michel Hufflen
- 15 La forma del testo umanistico: la classe `suftesi`
Ivan Valbusa
- 31 Una classe per comporre un dizionario
Claudio Beccari, Heinrich Fleck
- 41 L^AT_EX3: un nuovo gioco per i maghi e per diventarlo
Enrico Gregorio
- 48 Un pacchetto per il controllo del codice fiscale italiano
Claudio Beccari
- 53 Grafica ad oggetti con LuaT_EX
Roberto Giacomelli
- 72 MFLua: Instrumentation of METAFONT with Lua
Luigi Scarso
- 82 Gestione ‘quasi automatica’ dei valori numerici
in esempi di calcolo ed esercizi svolti
Agostino De Marco
- 95 Il pacchetto `esami` per la creazione di prove scritte
Grazia Messineo, Salvatore Vassallo
- 104 Test online di matematica: il Progetto M.In.E.R.Va
Grazia Messineo, Salvatore Vassallo
- 113 *Stemma codicum* con L^AT_EX
Matteo Fadini
- 123 Composizione di uno *Stemma codicum* con TikZ
Jerónimo Leal, Gianluca Pignalberi
- 131 Alcuni stralci sul Computer Bodoni
Claudio Vincoletto

