

Numero 21
Aprile 2016

ArsT_EXnica

Rivista italiana di T_EX e L^AT_EX

GUIT

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



G_UIT – Gruppo Utilizzatori Italiani di T_EX

ArsT_EXnica è la pubblicazione ufficiale del G_UIT

Comitato di Redazione

Claudio Beccari – *Direttore*

Roberto Giacomelli – *Comitato scientifico*

Enrico Gregorio – *Comitato scientifico*

Ivan Valbusa – *Comitato scientifico*

Lorena Rachele Badile, Renato Battistin,

Riccardo Campana, Massimo Caschili,

Gustavo Cevolani, Massimiliano Dominici,

Tommaso Gordini, Carlo Marmo,

Gianluca Pignalberi, Ottavio Rizzo,

Gianpaolo Ruocco, Enrico Spinielli,

Emiliano Vavassori

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

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

La rivista è aperta al contributo di tutti coloro che vogliano partecipare con un proprio articolo. Questo dovrà essere inviato alla redazione di ArsT_EXnica, per essere sottoposto alla valutazione dei revisori. 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 accetti, così come articoli riguardanti l'interazione con altre tecnologie correlate.

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

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

arstexnica@guitex.org.

Nota sul Copyright

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

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

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

Per maggiori informazioni:

<http://www.creativecommons.org>

Associarsi a G_UIT

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

Indirizzi

Gruppo Utilizzatori Italiani di T_EX

c/o Università degli Studi di Napoli Federico II

Dipartimento di Ingegneria Industriale

Via Claudio 21

80125 Napoli – Italia

<http://www.guitex.org>

guit@sssup.it

Redazione ArsT_EXnica:

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

arstexnica@guitex.org

Codice ISSN 1828-2369

Stampata in Italia

Napoli: 15 Aprile 2016

ArsT_EXnica

Rivista italiana di T_EX e L^AT_EX

Numero 21, Aprile 2016

Claudio Beccari	
Editoriale	3
C.V. Radhakrishnan, Hàn Thế Thành, Ross Moore, Peter Selinger	
Generazione di file PDF conformi alle norme PDF/X e PDF/A con pdfT _E X— pdfx	4
Clemens Niederberger	
La chimica con L ^A T _E X2 _ε Un esame dei pacchetti esistenti e delle possibilità	13
Luigi Scarso	
Latin Challenges	21
Roberto Giacomelli	
Primi esperimenti con node di LuaT _E X	31
Claudio Beccari	
La gestione delle lingue e delle loro varianti	45

Gruppo Utilizzatori Italiani di T_EX

Editoriale

Claudio Beccari

Ecco il numero 21 di *Ar_sT_EXnica*. Contiene pochi articoli ma densi di contenuti. In particolare due articoli sono apparsi in edizioni recenti di *TUGboat*; abbiamo creduto opportuno tradurli in italiano ritenendo che potessero essere interessanti per i soci del G_LT. Gli autori ci hanno subito concesso il loro consenso alla traduzione e noi siamo stati ben contenti di farlo.

Il primo articolo è stato scritto a quattro mani da quattro autori noti nel mondo di TUG. Si tratta di C.V. Radhakrisnan, caposquadra del gruppo di lavoro che si occupa dell'adattamento del sistema T_EX alla generazione di file archiviabili a lungo termine secondo la norma ISO 19005 del 2005; di Hàn Thé Thàn, il creatore del programma *pdf_{te}x*; di Ross Moore, il redattore della rivista elettronica *The Prac_TE_X Journal* e membro del gruppo di lavoro sul PDF/A; di Peter Selinger, anch'egli membro del gruppo di lavoro, è autore di numerosi importanti programmi fra i quali *Potrace* che ho usato abbondantemente quando creavo font.

Tornando all'articolo, essi descrivono le funzionalità del loro pacchetto *pdfx* che è lo strumento fondamentale per la creazione di file PDF/A (PDF Archiviabile) secondo i numerosi aggiornamenti dalla norma ISO che ho citato, e anche per produrre file PDF/X a norma per la stampa corretta. Gli autori si sono premurati di informarmi che le funzionalità sono descritte nell'articolo non sono ancora tutte disponibili con l'attuale aggiornamento della distribuzione T_EXLive 2015; sebbene io non le abbia collaudate tutte, mi pare che l'attuale pacchetto sia conforme a quanto descritto nell'articolo.

Il secondo articolo tradotto è di Clemens Niederberger; tratta delle composizioni della chimica con i programmi di tipocomposizione del sistema T_EX. Dalle singole equazioni di equilibrio, alle strutture complesse della chimica organica, dalla terminologia, ai simboli che vengono usati non solo per i componenti ma anche per i segnali di pericolo con le corrispondenti sigle. Un lavoro completissimo; della decina di pacchetti che descrive, più della metà li ha scritti lui, quindi è la persona più adatta per parlarne.

Luigi Scarso ci regala un suo lavoro di ricerca e di analisi costruttiva sulla composizione del latino, da quello arcaico via via fino a quello moderno ecclesiastico. La sua analisi storica dei primi reperti di scrittura latina sono estremamente interessanti e le sue soluzioni per far fare al programma *ConTeXt* le cose speciali necessarie per la scrittura

ra bustrofedica sono interessantissime. Scrittura bustrofedica? Sì la scrittura che usavano i nostri più antichi antenati (quelli capaci di scrivere) che scrivevano da sinistra a destra una riga e da destra a sinistra la riga successiva, per poi tornare alla scrittura da sinistra a destra; come quando si ara un campo andando alternatamente avanti e indietro con l'aratro. L'articolo è stato composto con *ConTeXt*, per cui ci potrà essere qualche piccola discrepanza rispetto alla veste grafica della rivista, che supporta ufficialmente solo L^AT_EX.

Roberto Giacomelli sta lavorando con LuaT_EX da tempo; finora si è servito essenzialmente di *lua_lat_ex* ricorrendo al linguaggio Lua solo per compiti speciali. Ora sta esplorando le possibilità di usarlo per comporre cose difficili senza ricorrere alle funzionalità di "Lua diretto"; ne scrive un buon tutorial, ma passo dopo passo conduce il lettore fino alla costruzione di un barcode con lo standard 39 con il quale si possono rappresentare stringhe alfanumeriche leggibili da qualunque lettore di barre basato sulla scansione mediante raggio laser.

Infine presento un mio articolo che descrive quanto è nascosto dietro *babel* e *polyglossia* per la gestione delle lingue; l'articolo vorrebbe invitare il lettore a conoscere meglio questa particolare funzione dei programmi di tipocomposizione del sistema T_EX ed esce poco dopo che nella sezione Documentazione del forum è apparsa la guida *L'Arte di scrivere in diverse lingue con (X_e)LaTeX* scritta da Tommaso Gordini e da me. Chiaramente questa guida è decisamente più approfondita, ma spero che l'articolo che compare in questo numero di *Ar_sT_EXnica* stimoli anche i lettori curiosi a documentarsi meglio nella guida citata.

Ringrazio moltissimo i membri del comitato di redazione, i reviewer che hanno esaminato e valutato gli articoli qui pubblicati, i revisori editoriali e i correttori che sono intervenuti sugli articoli in italiano e in inglese per migliorarne il testo. Il lavoro di queste persone è encomiabile; e senza di loro la qualità degli articoli, al di là della competenza degli autori, non sarebbe quella che ognuno di noi può constatare leggendo questi nuovi cinque articoli.

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

Generazione di file PDF conformi alle norme PDF/X e PDF/A con pdfTeX—pdfx*

C.V. Radhakrishnan, Hàn Thế Thành, Ross Moore, Peter Selinger

1 Introduzione

Il pacchetto pdfx (`pdfx.sty`) al momento permette di compilare documenti conformi alle norme PDF/X e PDF/A usando pdfTeX, rispettando alcune varianti di questi standard. Sono disponibili anche le funzionalità per gli standard PDF/E e PDF/VT; la lista completa è nel paragrafo 2.1. Con l'espressione “permette di compilare” intendiamo che il pacchetto mette a disposizione i mezzi corretti e sufficienti per far sì che un documento possa essere dichiarato conforme a una particolare variante di PDF (PDF/X, PDF/A, PDF/E, PDF/VT, eccetera) insieme ad un particolare livello di conformità. Il pacchetto consente anche di inserire i metadati e il profilo di colore secondo le specifiche della particolare variante PDF scelta.

Gli elementi che formano i metadati, la maggior parte dei quali alla fine vanno inseriti in XML usando la codifica UTF-8, sono forniti mediante un file col nome `\jobname.xmpdata` per ciascun documento L^AT_EX. Senza questo file, che fornisce alcune informazioni necessarie e diverse informazioni accessorie, non si può ottenere un file PDF completamente conforme ai requisiti. Il PDF può essere creato con l'aspetto corretto di ogni pagina, ma non passerà i controlli di validazione. Il paragrafo 2.2 spiega come costruire questo file.

Quello che questo pacchetto *non fa* è quello di controllare tutti i dettagli della struttura del documento e del suo contenuto che può essere richiesto (o proibito) secondo la specifica variante PDF. Per esempio, PDF/VT (ISO 16612) richiede che il documento sia ben strutturato mediante sezioni del tipo “Form XObject” etichettate come `/DPart`. Nella stessa maniera PDF/A-1a (e anche 2a e 3a) (ISO 19005-1; ISO 19005-2; ISO 19005-3)) richiedono che il file PDF sia completamente “Tagged PDF”, inclusa una struttura di marcature che qualifica completamente i contenuti del documento. Questo non si può ancora ottenere con la versione corrente di pdfTeX, come lo si trova normalmente nelle distribuzioni del sistema T_EX. Perciò, mentre questo pacchetto è in grado di fornire dati sufficienti per soddisfare la dichiarazione, i metadati e la gestione dei font per tutte le varianti PDF/A, non è ancora in grado di produrre in ogni circostanza

file completamente conformi. Però con altro software esterno a pdfTeX che *sia* capace di produrre file di tipo “Tagged PDF”, questo pacchetto può essere usato come parte del normale ciclo di lavoro per produrre file completamente compatibili.

1.1 Gli standard PDF

PDF/X e PDF/A sono termini generici per denotare diversi standard ISO (ISO 19005-1; ISO 19005-2; ISO 19005-3; ISO 15930-1; ISO 15930-3; ISO 15930-4; ISO 15930-7; ISO 15930-8) che definiscono sottoinsiemi diversi dello standard PDF (ADOBE SYSTEMS INC., 2006; ISO 32000-1; ISO 32000-2). Lo scopo di PDF/X è quello di facilitare lo scambio di elementi grafici tra il creatore del documento e il mezzo di stampa e perciò ha tutti i requisiti legati al processo di stampa. Per esempio con PDF/X tutti i font devono essere incorporati e tutte le immagini devono essere in colore CMYK oppure in colori spot. PDF/X-2 e PDF/X-3 accettano colori RGB calibrati e colori CIELAB oltre alle altre specifiche di PDF/X. Dal 2005 sono emerse altre varianti di PDF/X, visto che altri effetti (come gli strati (*layering*) e le trasparenze) sono gestiti dallo stesso standard PDF. La serie completa di versioni e di requisiti di conformità è discusso nel paragrafo 2.1.4.

PDF/A definisce un profilo per archiviare documenti PDF, che assicuri che i documenti possano essere riprodotti esattamente negli anni futuri. Un elemento chiave per ottenere questo scopo è che un documento PDF/A sia completamente autocontenuto. Tutta l'informazione necessaria per mostrare sempre in modo identico il documento deve essere incorporata nel file. Un documento PDF/A non deve appoggiarsi assolutamente su informazioni esterne al documento stesso. Altre restrizioni richiedono che il file non contenga effetti multimediali (come audio e video), né JavaScript né crittografia. L'inclusione dei font e la specificazione del profilo di colore è obbligatoria per un file PDF/A, come lo sono i metadati da inserire secondo uno standard ben definito. Versioni successive consentono la compressione delle immagini e che vi siano allegati altri file.

PDF/E è uno standard ISO (ISO 24517) finalizzato ai documenti usati nel processo di lavoro ingegneristico. PDF/VT (ISO 16612) è finalizzato alla produzione di documenti in grandi quantità personalizzati per esempio per stampare le bollette

*Col permesso degli autori questo loro articolo apparso su *TUGboat* 36(2015):2 è stato tradotto da C. Beccari; eventuali errori o travisamenti di significato sono dovuti solamente al traduttore. *N.d.T.*

delle utenze. PDF/UA (Universal Accessibility) è in fase di sviluppo come un nuovo standard (ISO 14289; ISO PDFUA) per contenere aspetti tecnologici di assistenza (ai disabili) che incorporano le linee guida (WCAG) per l'accesso in rete ai documenti elettronici. Nel futuro potrebbe venire sviluppato PDF/H per registrazioni relative alla salute e per documenti sanitari. Si possono immaginare ulteriori applicazioni. Le dichiarazioni e i metadati necessari sono disponibili per i primi due tipi. Le altre sono oggetto di lavoro futuro; si può prevedere che nei prossimi anni possano venire prodotte versioni modificate ed estese di questo pacchetto.

2 L'uso

Il pacchetto viene caricato col solito comando:

```
\usepackage[<opzioni>]{pdfx}
```

Le opzioni disponibili sono descritte qui di seguito¹.

2.1 Opzioni

2.1.1 Opzioni per PDF/A

PDF/A è uno standard ISO previsto per l'archiviazione di documenti elettronici nel lungo periodo. Perciò le proprietà di essere autocontenuto e sempre esattamente riproducibile sono tenute in particolare attenzione; richiede inoltre metadati leggibili direttamente su ogni macchina. Lo standard PDF/A ha tre livelli di conformità "a", "b" e "u". Il livello "a" è quello che richiede i requisiti più stringenti e non è ancora completamente sviluppato nel pacchetto pdfx. Il livello "u" ha i medesimi requisiti del livello "b", con il requisito ulteriore che tutti i font usati nel documento siano codificati in UNICODE. Va notato che il pacchetto pdfx produce la mappatura UNICODE anche nei file di livello "b". Lo standard ha inoltre tre diverse versioni, 1, 2, e 3, che sono state definite rispettivamente nel 2005, 2011 e nel 2012. Le versioni precedenti contengono un sottoinsieme delle prerogative delle versioni successive così per garantire la massima portabilità è preferibile usare una versione di numero più basso. Non esiste un livello di conformità "u" nella versione 1. Per gli usi più tipici del PDF/A è sufficiente usare il PDF/A-1b. Le opzioni sono le seguenti:

- **a-1a**: genera un PDF/A-1a; sperimentale, non ancora completamente implementato;
- **a-1b**: genera un PDF/A-1b;
- **a-2a**: genera un PDF/A-2a; sperimentale, non ancora completamente implementato;
- **a-2b**: genera un PDF/A-2b;
- **a-2u**: genera un PDF/A-2u;
- **a-3a**: genera un PDF/A-3a; sperimentale, non ancora completamente implementato;
- **a-3b**: genera un PDF/A-3b;
- **a-3u**: genera un PDF/A-3u.

Con l'espressione: 'sperimentale, non ancora completamente implementato', intendiamo sostanzialmente che la struttura 'Tagged PDF' non è gestita con questo pacchetto. Usando altro software basato su pdfTeX che *sia* capace di produrre questa marcatura completa, si possono davvero produrre documenti conformi².

2.1.2 Opzioni per PDF/E

PDF/E è uno standard ISO destinato ai documenti usati nel processo produttivo ingegneristico. A tutt'oggi c'è una sola versione dello standard PDF/E e si chiama PDF/E-1. L'opzione

- **e-1**: genera un PDF/E-1.

2.1.3 Opzioni per PDF/VT

PDF/VT è uno standard ISO destinato ai documenti in un formato di scambio per la stampa con dati variabili al fini di produrre documenti legati a certe forme di transazioni ed è una estensione dello standard PDF/X-4. Lo standard PDF/VT prevede tre livelli di conformità; il livello 1 è per file singoli; il livello 2 è per file multipli; e il livello 3 è per la consegna dei documenti in modo streamed. Al momento nessuno dei livelli di conformità PDF/VT è completamente ottenuto con il pacchetto pdfx. Le opzioni sono le seguenti:

- **vt-1**: genera un PDF/VT-1; sperimentale, non ancora completamente implementato;
- **vt-2**: genera un PDF/VT-2; sperimentale, non ancora completamente implementato;
- **vt-3**: genera un PDF/VT-3; sperimentale, non ancora completamente implementato.

Con l'espressione: 'sperimentale, non ancora completamente implementato', intendiamo sostanzialmente che la struttura del documento in sezioni 'DPart' di 'Form XObjects', non è gestita da questo

2. Essendo in contatto con il team che si occupa di queste funzionalità per il sistema TeX, sono al corrente che la futura versione 1.50 di pdfTeX (non ancora distribuita) potrà produrre file di tipo Tagged PDF. (*N.d.T.*)

1. In Europa è necessario caricare il pacchetto dopo aver specificato la codifica in entrata, chiamando con l'opportuna opzione il pacchetto `inputenc`, e prima di chiamare il pacchetto `hyperref` senza opzioni; se invece di pdfLaTeX si usa LuaLaTeX bisogna comunque caricare il pacchetto prima di `hyperref`, sempre senza opzioni; se si vuole personalizzare il comportamento di `hyperref`, bisogna usare il comando `\hypersetup`. Con LuaLaTeX, apportando un piccolo patch a pdfx e usando solo font OpenType, si riesce ad ottenere un file conforme alle norme anche se la documentazione di pdfx, al momento di tradurre questo file, non dice nulla in proposito.. (*N.d.T.*)

pacchetto. Questo è già possibile con il software pdfTeX, ma non nella maniera che si presta per una realizzazione completamente automatica, perché bisogna conoscere l'*Object number* di alcuni costrutti interni del file PDF. Tutti gli altri aspetti — la dichiarazione per il dizionario PDF `/Info`, i metadati e il profilo di colore — delle varianti PDF/VT sono gestite correttamente.

2.1.4 Opzioni per PDF/X

Lo standard ISO PDF/X è previsto per l'interscambio di file con contenuto grafico. Esso sottolinea gli aspetti che riguardano la stampa, come il fatto che i font siano incorporati insieme ai profili di colore. Lo standard PDF/X ha diversi livelli di conformità e di varianti. Le varianti di base sono X-1, X-1a, X-3, X-4 e X-5. (La versione riveduta e corretta X-2 è stata pubblicata nel 2003, ma è stata ritirata nel 2011, sostanzialmente per mancanza di interesse nell'usarla.) Lo standard PDF/X-1a è presente nelle revisioni del 2002 e del 2003, lo standard PDF/X-3 esiste nelle revisioni del 2001 e del 2003, gli standard PDF/X-4 e PDF/X1-5 esistono nelle revisioni del 2008 e del 2010. Alcune di queste varianti hanno una versione 'p' che permette l'uso di un profilo di colore esterno (invece di usare un profilo incluso), e/o una versione 'g' che consente l'uso di file grafici esterni. Inoltre PDF/X-5 ha una versione 'n' che estende lo standard PDF/X-4p, permettendo ulteriori spazi di colore oltre alla scala di grigi, lo spazio RGB e quello CMYK. Per molti usi tipici di PDF/X è sufficiente usare la variante PDF/X-1a.

- **x-1**: genera un file PDF/X-1;
- **x-1a**: genera un file PDF/X-1a; le ulteriori opzioni **x-1a1** e **x-1a3** sono disponibili per specificare esplicitamente le varianti approvate nel 2001 o nel 2003;
- **x-3**: genera un file PDF/X-3; le ulteriori opzioni **x-302** e **x-303** sono disponibili per specificare esplicitamente le varianti approvate nel 2002 o nel 2003;
- **x-4**: genera un file PDF/X-4; le ulteriori opzioni **x-408** e **x-410** sono disponibili per specificare esplicitamente le varianti approvate nel 2008 o nel 2010;
- **x-4p**: genera un file PDF/X-4p; le ulteriori opzioni **x-4p08** e **x-4p10** sono disponibili per specificare esplicitamente le varianti approvate nel 2008 o nel 2010;
- **x-5g**: genera un file PDF/X-5g; le ulteriori opzioni **x-5g08** e **x-5g10** sono disponibili per specificare esplicitamente le varianti approvate nel 2008 o nel 2010;
- **x-5n**: genera un file PDF/X-5n; le ulteriori opzioni **x-5n08** e **x-5n10** sono disponibili per

specificare esplicitamente le varianti approvate nel 2008 o nel 2010; sperimentale, non ancora completamente implementato.

- **x-5pg**: genera un file PDF/X-5pg; le ulteriori opzioni **x-5pg08** e **x-5pg10** sono disponibili per specificare esplicitamente le varianti approvate nel 2008 o nel 2010;

2.1.5 Altre opzioni

Le seguenti opzioni per il pacchetto pdfx sono sperimentali e normalmente non dovrebbero venire usate.

- **useBOM**: genera esplicitamente un marcatore di ordine di byte nella codifica UTF-8 (*UTF-8 byte-order marker*) nei metadati XMP inclusi e rende il pacchetto XMP scrivibile. Nessuna di queste funzionalità è richiesta per lo standard PDF/A, ma sembra che ci siano alcuni validatori di file PDF/A che li richiedono. Nota: l'implementazione di questa funzionalità è sperimentale e potrebbe cessare di funzionare con i futuri aggiornamenti del pacchetto `xmpincl`.
- **noBOM**: non genera il *byte-order marker*; questa è l'impostazione di default.
- **pdf13**: usa PDF nella versione 1.3, scavalcando la versione specificata dallo standard richiesto. Questo può impedire la validazione del file PDF.
- **pdf14**: usa PDF nella versione 1.4, scavalcando la versione specificata dallo standard richiesto. Questo può impedire la validazione del file PDF.
- **pdf15**: usa PDF nella versione 1.5, scavalcando la versione specificata dallo standard richiesto. Questo può impedire la validazione del file PDF.
- **pdf16**: usa PDF nella versione 1.6, scavalcando la versione specificata dallo standard richiesto. Questo può impedire la validazione del file PDF.
- **pdf17**: usa PDF nella versione 1.7, scavalcando la versione specificata dallo standard richiesto. Questo può impedire la validazione del file PDF.

2.2 Il file dei metadati

Come detto precedentemente, i documenti conformi allo standard PDF/A richiedono che vi siano inclusi alcuni metadati composti secondo precise direttive (DUBLIN CORE, 2010). Il pacchetto pdfx si aspetta che i metadati siano forniti tramite un file speciale dal nome `\jobname.xmpdata`. Qui `\jobname` è il nome del file `.tex` principale per

la composizione del documento. Per esempio se il vostro file principale si chiama `main.tex`, allora il file dei metadati deve chiamarsi `main.xmpdata`. Nessuno dei singoli campi dei metadati è obbligatorio, ma per la maggior parte dei documenti è ragionevole definire almeno il titolo e l'autore. Qui c'è un esempio di un breve file di metadati:

```
\Title{Storia della cottura al forno}
\Author{U. Fornaio\sep A. Pastaio}
\Keywords{Biscotti\sep crescenti}
\Publisher{Casa editrice I Forni}
```

Si noti che una lista di più autori e di più parole-chiave ha gli elementi separati con il comando `\sep`. L'uso della macro `\sep` è consentito solo nei campi `\Author`, `\Keywords` e `\Publisher`.

Dopo l'elaborazione dei metadati, la cartella di lavoro contiene un file del tipo `pdfa.xmpi` o `pdfx.xmpi` a seconda della variante PDF richiesta. Questo file costituisce il pacchetto dei metadati. Può essere controllato per verificarne la validità mediante un validatore on-line, come ad esempio: <http://www.pdflib.com/knowledge-base/xmp-metadata/free-xmp-validator>

2.3 Elenco dei campi per i metadati

Qui viene esposta la lista completa dei metadati attualmente definibili dall'utente corredati del loro significato. Se ne potranno aggiungere altri in futuro. I comandi che seguono possono essere usati solo nei file `.xmpdata`.

2.3.1 Informazioni generali

- `\Author`: contiene il nome e il cognome dell'autore del documento. Più autori vanno separati con `\sep`. Il nome può essere abbreviato con l'iniziale puntata.
- `\Title`: contiene il titolo del documento.
- `\Keywords`: contiene l'elenco delle parole chiave separate con `\sep`.
- `\Subject`: contiene un breve sommario del documento.
- `\Publisher`: Contiene la ragione sociale della casa editrice o dell'organizzazione che ha prodotto il documento.

2.3.2 Informazioni sul copyright

- `\Copyright`: contiene la dichiarazione del copyright.
- `\CopyrightURL`: contiene l'indirizzo web che descrive il possessore del copyright e/o i diritti di copyright relativi al documento.
- `\Copyrighted`: contiene "True" se il documento è soggetto a copyright o "False" altrimenti. Questo metadato è impostato automaticamente a "True" se sono stati specificati i campi

`\Copyright` o `\CopyrightURL`, ma può essere scavalcato. Per esempio, se la dichiarazione di copyright è "Di pubblico dominio", questo campo dovrebbe essere impostato a "False".

2.3.3 Informazioni sulla pubblicazione

- `\PublicationType`: contiene il nome del tipo di pubblicazione. Se questo campo viene specificato, esso può contenere solo uno dei nomi seguenti: book, catalog, feed, journal, magazine, manual, newsletter, pamphlet. Questo campo è automaticamente impostato a "journal" se si specifica il campo `\Journaltitle`.
- `\Journaltitle`: contiene il titolo della rivista dove è stato pubblicato il documento.
- `\Volume`: contiene il numero del volume della rivista.
- `\Issue`: contiene il numero della rivista.
- `\Firstpage`: contiene il numero della prima pagina dove inizia il documento pubblicato.
- `\Lastpage`: contiene il numero dell'ultima pagina dove finisce il documento pubblicato.
- `\Doi`: contiene il Digital Object Identifier (DOI) del documento senza il prefisso "doi:".
- `\CoverDisplayDate`: contiene la data che compare sulla copertina del fascicolo della rivista come data da leggere da parte delle persone.
- `\CoverDate`: contiene la data che compare sulla copertina del fascicolo della rivista in un formato adatto per essere memorizzato in un campo di un database avente la tipologia "date".

2.4 Simboli consentiti nei metadati

Nei metadati sono usabili tutti i caratteri ASCII, tranne `\`, `char123`, `}` e `%`, i quali rappresentano se stessi. Inoltre sono usabili tutti i caratteri e i simboli UNICODE del piano di base (cioè fino al code point U+FFFF) ma devono essere rappresentati con la transcodifica UTF-8. (Attualmente non sono accettabili nei metadati codifiche diverse da UTF-8). Le sequenze di spazi vengono contratti in un solo spazio. Lo spazio dopo una macro come `\copyright`, `\backslash` o `\sep` viene ignorato. In più si può usare questo mark-up:

- `\_`: uno spazio reale, per esempio dopo una macro;
- `\%`: un % letterale;
- `\{`: una { letterale;
- `\}`: una } letterale;

- `\backslash`: un `\` letterale
- `\copyright`: il simbolo di copyright ©.

La macro `\sep` è consentita solo negli argomenti di `\Author`, `\Keywords` e `\Publisher`. Il suo scopo è proprio quello di separare una molteplicità di autori, parole chiave, eccetera. Però, ai fini della validazione, gli autori e le parole chiave non devono essere veramente separati, ma il pacchetto ne tiene conto anche quando si usa `\sep`.

Risulta che si possono usare anche altre macro T_EX, purché l'autore sia molto attento ad evitare macro T_EX troppo complicate o sviluppi di macro L^AT_EX nei comandi di base o primitive diverse da caratteri; in sostanza sono accettabili solo gli accenti, le macro per caratteri latini speciali, e semplici sostituzioni testuali, talvolta con un semplice argomento. Viene fornita la macro speciale

```
\cs{pdfxEnableCommands{...}
```

per aiutare a risolvere situazioni difficili che potrebbero presentarsi.

Per esempio, `\pdfxEnableCommands` è necessaria per il nome di uno degli autori Hàn Thế Thành, a causa della ‘e’ con due accenti ‘é’. Si è soliti definire una macro

```
\def\thanh{H\‘an Th\’{\~e} \Th\‘anh}
```

Nelle versioni precedenti del pacchetto pdfx l'uso di questa macro all'interno del file `.xmpdata`, per esempio nella dichiarazione di copyright eseguita con `\Copyright`, produceva lo sviluppo delle macro per gli accenti in termini di comandi primitivi, del tipo di

```
H\unhbox \voidb@x \bgroup
\let unhbox \voidb@x \setbox ...
```

e così di seguito per diverse righe. Questo evidentemente è fuori luogo fra i metadati XMP. Per aggirare questo inconveniente si sarebbero potute usare delle definizioni semplificate:

```
\pdfxEnableCommands{
\def\‘#1{#1~cc~80}
\def\’#1{#1~cc~81}
\def\~#1{#1~cc~82}}
```

dove le sequenze `~cc~80`, `~cc~81`, `~cc~82`, permettono a T_EX di inserire i byte in codifica UTF-8 per i caratteri definiti “combining accent”. Questo funzionerebbe per i metadata che compaiono solo nel pacchetto XMP. Tuttavia non è sufficiente per il campo `/Author` del file PDF, che deve corrispondere esattamente all'elemento dei metadati `dc:creator`. Quello che serve, invece è:

```
\pdfxEnableCommands{
\def\thanh{H~c3~a0n Th~eee Th~c3~a0nh}
\def~eee{~c3~aa~cc~81 }}
```

o la stessa cosa con ‘à’ inserito direttamente in UFF-8 al posto di `~cc~a0` ed ‘è’ al posto di `~c3~aa`. Il motivo per fare ciò nasce dal comando `\pdfstringdef` che costruisce direttamente le lettere latine come singoli caratteri à ed ê, senza ricorrere ai “combining accents”, almeno quando è possibile. Se i metadati non contengono esattamente le stesse cose, nonostante la stessa normalizzazione UNICODE, la validazione fallisce.

Con l'ultima versione del pacchetto pdfx, queste difficoltà sono state superate almeno per le lingue dell'Europa occidentale basate sull'alfabeto latino. La codifica d'entrata usata durante la lettura del file `.xmpdata` ora include l'interpretazione dei soliti comandi T_EX per gli accenti al fine di produrre le giuste sequenze di byte per la codifica UTF-8. Stiamo lavorando per estendere questa codifica d'entrata per altre lingue scritte con altri alfabeti (per esempio, latino esteso, cirillico, greco, eccetera). Una parte significativa dei caratteri del piano di base Unicode può essere trattato nello stesso modo. Si potrebbero fornire anche i moduli per i caratteri CJK (cinese, giapponese, coreano), per i simboli matematici, ed altro. Questo però diventerebbe un forte carico per la memoria, quindi sono ancora necessari molti test significativi prima che queste estensioni possano far parte del pacchetto pdfx.

2.5 Profili di colore

La maggior parte dei documenti conformi agli standard PDF richiedono che un *profilo di colore* sia inserito nel file. In breve, questo profilo determina con precisione come i colori usati nel documento abbiano lo stesso aspetto anche quando vengano riprodotti su mezzi fisici diversi. Questo deve assicurare che il documento mantenga esattamente il suo aspetto anche quando viene stampato con stampanti diverse, con tipi diversi di carta, o vengano mostrati su schermi diversi. L'inclusione di un profilo di colore è necessario affinché il documento sia autocontenuto.

Siccome la maggior parte degli utenti di L^AT_EX non sono dei professionisti grafici e spesso non sono particolarmente attenti alle sfumature di colore, il pacchetto pdfx è provvisto di un certo numero di profili di default che vengono inclusi nel documento finale se non si specifica qualcosa d'altro. Perciò l'utente medio non ha di che preoccuparsi per ciò che riguarda il colore. Gli utenti che desiderano usare un profilo di colore specifico, lo possono fare inserendo fra i metadati un comando `\setRGBcolorprofile` oppure `\setCMYKcolorprofile` nel file `.xmpdata`. Si noti che gli standard PDF/A e PDF/E accettano solo un profilo di colore RGB, mentre gli standard PDF/X e PDF/VT richiedono un profilo di colore CNYK³. Si usino quindi i comandi per specificare

3. salvo alcune varianti che accettano anche altri profili. (N.d.T.)

TABELLA 1: Comandi per diversi profili di colore

Comando	Nome del profilo
<code>\FOOGRAXXXIX</code>	Coated FOOGRA39 (ISO 12647-2:2004)
<code>\SWOPCGATSI</code>	U.S. Web Coated (SWOP) v2
<code>\JapanColorMMICoated</code>	Japan Color 2001 Coated
<code>\JapanColorMMIUncoated</code>	Japan Color 2001 Uncoated
<code>\JapanColorMMIINewspaper</code>	Japan Color 2002 Newspaper
<code>\JapanWebCoatedAd</code>	Japan Web Coated (Ad)
<code>\CoatedGRACoL</code>	Coated GRACoL 2006 (ISO 12647-2:2004)
<code>\SNAPCGATSII</code>	CGATS TR 002
<code>\SWOPCGATSIII</code>	CGATS TR 003
<code>\SWOPCGATSV</code>	CGATS TR 005
<code>\ISOWebCoated</code>	Web Coated FOGRA28 (ISO 12647-2:2004)
<code>\ISOCoatedECI</code>	ISO Coated v2 (ECI)
<code>\CoatedFOGRA</code>	Coated FOGRA27 (ISO 12647-2:2004)
<code>\WebcoatedFOGRA</code>	Web Coated FOGRA28 (ISO 12647-2:2004)
<code>\UncoatedFOGRA</code>	Uncoated FOGRA29 (ISO 12647-2:2004)
<code>\IFRAXXVI</code>	ISOnewspaper26v4 ISO/DIS 12647-3:2004
<code>\IFRAXXX</code>	ISOnewspaper30v4 ISO/DIS 12647-3:2004

i profili di colore RGB o CMYK secondo la sintassi seguente:

```
\setRGBcolorprofile<nome del file>
<identificatore><stringa info><URL di
registrazione>
```

```
\setCMYKcolorprofile<nome del file>
<output intent><identificatore><URL di
registrazione>
```

Negli argomenti di queste macro i caratteri <, >, &, ^, _, #, \$, ~, rappresentano se stessi ma per il segno % bisogna usare il comando \%. Le impostazioni di default sono le seguenti:

```
\setRGBcolorprofile
{ sRGB_iEC61966-2-1_black_scaled.icc }
{ sRGB_iEC61966-2-1_black_scaled }
{ sRGB_iEC61966-2-1 with black scaling }
{ http://www.color.org }
```

```
\setCMYKcolorprofile
{ coated_FOGRA39L_arg1.icc }
{ Coated FOGRA39 }
{ FOGRA (ISO Coated v2 300% (ECI)) }
{ http://www.argyllcms.com/ }
```

Alcuni altri profili di colore si possono scaricare dall'International Color Consortium; si veda nel sito <http://www.color.org/iccprofile.xalter>.

In alternativa molti profili di colore sono allegati a molte applicazioni software della Adobe; questi diventano usabili anche con software non di origine Adobe. Ora il pacchetto pdfx include il codice per l'immissione diretta di questi profili nei documenti PDF, o per specificarli come profili esterni con le varianti PDF/X4p e PDF/X-5pg. Due file `AdobeColorProfiles.tex` e

`AdobeExternalProfiles.tex` sono contenuti nella distribuzione del pacchetto pdfx. Il secondo file serve con le varianti PDF/X-4p e PDF/X5pg che non richiedono l'immissione del profilo di colore nel file del documento, mentre il primo serve con le altre varianti di PDF/X. Entrambi definiscono comandi per usare quei profili di colore come indicato nella tabella 1.

Al momento di scrivere solo i primi sei di quei profili di colore esterni (cioè usati con PDF/X-4p o PDF/X-5pg) portano a file PDF che risultano validati usando la versione corrente di Preflight del programma Adobe Acrobat Pro. Non è chiaro se gli altri profili (inclusi `\IFRAXXVI` e `\IFRAXXX`) falliscono a causa di dati non corretti o per problemi connessi con il software di validazione. Tutti, tranne gli ultimi due possono essere usati per profili inclusi validi, se i file corrispondenti sono effettivamente accessibili per l'inclusione. La macro seguente può essere usata per fissare il percorso (assoluto o relativo) nel sistema operativo locale per accedere ai file dei profili di colore.

```
\pdfxSetColorProfileDir
{ <percorso ai profili di colore Adobe> }
```

2.6 Note sulla rappresentazione interna dei metadati

All'interno del file PDF i metadati sono inseriti in due punti: alcuni dati vanno a finire nel dizionario /Info nativo del PDF, mentre alcuni altri dati vanno a finire in un pacchetto XMP memorizzato in una posizione separata all'interno del file. XMP è il formato dell'Extensible Metadata Platform dell'Adobe, ed è un formato basato su l'XML. Si veda l'Adobe Development Center (<http://www.adobe.com/devnet/xmp>) per maggiori informazioni in merito all'XMP. È anche

disponibile un XMP Toolkit SDK, usabile su GNU/Linux, Macintosh e Windows rilasciato con una licenza BSD modificata.

Alcuni dei metadati come l'autore, il titolo e le parole chiave sono memorizzati sia nel dizionario /Info sia nel pacchetto XMP. Per la validazione di conformità con gli standard le due copie dei dati devono essere identiche. Di ciò si occupa il pacchetto pdfx.

In linea di principio gli utenti possono ricorrere ad altri mezzi per includere un file XMP. In questo caso gli utenti dovrebbero creare un file pdfa.xmp o pdfx.xmp (o altro, a seconda del tipo di PDF) che contengono i dati predefiniti. Ma va osservato che questo modo di procedere si presta a facili errori e non lo si raccomanda alla maggior parte degli utenti. Se c'è un campo di metadati particolare di cui l'utente ha bisogno ma che non è ancora disponibile, l'utente è invitato a rivolgersi agli autori.

pdfx si serve del pacchetto xmpincl per inserire i dati XMP nel pdf. La documentazione del pacchetto xmpincl può essere utile per capire il meccanismo dei metadati XMP.

2.7 Tutorial e note tecniche

Un file di iniziazione con istruzioni passo-passo per generare file PDF/A si trova all'indirizzo:

<http://wwwmathstat.dal.ca/~selinger/pdfa>

Alcune note tecniche in merito ad alcuni problemi che gli autori hanno incontrato nel produrre e nel generare documenti PDF/A conformi allo standard si trovano all'indirizzo:

http://support.river-valley.com/wiki/index.php?title=Generating_PDF/A_compliant_PDFs_from_pdftex

3 Installazione

Il pacchetto pdfx è scaricabile da CTAN attraverso <http://ctan.org/pkg/pdfx>. Esso è già incluso nelle distribuzioni aggiornate e complete del sistema T_EX come T_EXLive e MiK_TE_X.

3.1 Limiti e dipendenze

pdfx.sty lavora con pdfT_EX e anche con LuaT_EX⁴. Inoltre esso dipende dai seguenti pacchetti:

4. Nella documentazione del pacchetto pdfx c'è scritto esplicitamente che esso lavora solo con pdfT_EX (PDFL_AT_EX); in effetti lanciando la compilazione con LuaL_AT_EX su un file predisposto come indicato, *mutatis mutandis*, per lavorare con LuaL_AT_EX, la compilazione non va a buon fine, perché il motore LuaT_EX (almeno al momento attuale in cui si sta eseguendo questa traduzione) manca di un comando nativo di pdfT_EX; tuttavia si può richiamare il pacchetto di Heiko Oberdiek `pdftexcmds` che fornisce a LuaT_EX i comandi specifici di pdfT_EX ma non (più) inclusi in LuaT_EX, e la compilazione va a buon fine; il file può risultare validabile se sono verificate tutte le clausole richieste dallo standard.. (N.d.T.)

1. xmpincl per l'inserimento dei metadati nel file PDF;
2. hyperref per eseguire i collegamenti ipertestuali, creare i bookmark, eccetera;
3. glyphtounicode.tex per mappare i nomi dei glifi sui nomi UNICODE.

3.2 I file inclusi

I seguenti file fanno parte del pacchetto; alcuni sono generati da pdfx secondo le indicazioni di Makefile⁵.

3.2.1 I file del pacchetto

- pdfx.sty: è il file principale del pacchetto, (estratto da pdfx.dtx).
- pdfa.xmp: modello di file .xmp per produrre file PDF/A.
- pdfx.xmp: modello di file .xmp per produrre file PDF/X.
- 8bit.def: codifica d'entrata speciale.
- l8uenc.def: dichiarazione di macro per la codifica d'entrata.
- glyphtounicode-cmr.tex: mappa i nomi dei glifi dei font Computer Modern e di altri font specifici di T_EX sui nomi UNICODE.
- coated_F00GRA39L_arg1.icc: profilo di colore CMYK liberamente distribuibile.
- sRGB_IEC61966-2-1_black_scaled.icc: profilo di colore RGB, liberamente distribuibile.
- ICC_LICENSE.txt: licenza per i profili di colore.
- AdobeColorProfiles.tex: macro per l'inclusione dei profili di colore della Adobe.
- AdobeExternalProfiles.tex: macro per usare profili di colore esterni.

5. L'utente che si serva di T_EXLive o di MiK_TE_X non deve preoccuparsi di Makefile perché quelle distribuzioni del sistema T_EX provvedono automaticamente all'installazione del pacchetto pdfx. (N.d.T.)

3.2.2 Documentazione

- **README**: è la solita informazione generale in formato `.txt` anche se manca di questa estensione.
- **manifest.txt**: elenco dei file del pacchetto.
- **sample.tex** e **sample.xmpdata**: un esempio di file con un esempio di file contenente i metadati.
- **small12e-pdfx.tex**, **small12e-pdfx.xmpdata**: un altro esempio con i corrispondenti metadati.

3.2.3 I file sorgente

- **src/pdfx.dtx**: “Documented T_EX file” contenente il codice e la documentazione.
- **src/pdfx.ins**: il file batch per l’installazione.
- **src/rvdtx.sty**: usato dal file **pdfx.dtx**.
- **Makefile**: uno script Makefile per produrre la documentazione.

3.3 Licenza e contatti

Il pacchetto è soggetto alla L^AT_EX Project Public Licence. Segnalazioni di errori, suggerimenti, richieste di ulteriori funzionalità, eccetera, possono essere inviate a qualunque degli autori indicati alla fine dell’articolo.

Riferimenti bibliografici

ADOBE SYSTEMS INC. (2006). «PDF Reference 1.7». URL http://www.adobe.com/devnet/pdf/pdf_reference.html.

DUBLIN CORE (2010). «Core Metadata Element Set, Version 1.1». URL <http://dublincore.org/documents/dces/>.

ISO 14289 (2012). «ISO 14289-1:2012; Document management applications — Electronic document file format enhancement for accessibility — Part 1: Use of ISO 32000-1 (PDFUA-1)». URL http://www.iso.org/iso/catalogue_detail?csnumber=54564. Technical Committee ISO/TC 171/SC2 Riveduto col nome ISO 14289-1:2104 all’indirizzo web http://www.iso.org.iso/catalogue_details?csnumber=64599.

ISO 15930-1 (2001). «ISO 15930-1:2001; Graphic Technology — Prepress digital data exchange — Use of PDF — Part 1: Complete exchange using CMYK data (PDF/X-1 and PDF/X-1a)». URL http://www.iso.org/iso/catalogue_details.htm?csnumber=29061. Technical Committee ISO/TC 130.

ISO 15930-7 (2010). «ISO 15930-7:2010; Graphic technology — Prepress digital data exchange — Use of PDF — Part 7: Complete exchange of printing data (PDF/X-4) and partial exchange of printing data with external colour profile reference (PDF/X-4p) using PDF 1.6». URL http://www.iso.org/iso/catalogue_detail.htm?csnumber=55843. Technical Committee ISO/TC 130.

ISO 16612 (2005). «ISO 16612-2:2010; Graphic Technology — Variable data exchange — Part 2: Using PDF/X-4 and PDF/X-5 (PDFVT-1 and PDFVT-2)». URL http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=38013. Technical Committee ISO/TC 130.

ISO 19005-1 (2005). «ISO 19005-1:200; Document Management — Electronic document file format for long term preservation — Part 1: Use of PDF 1.4 (PDF/A-1)». URL http://www.iso.org/iso/catalogue_detail?csnumber=38920. Technical Committee ISO/TC171/SC2. Revisions via corrigenda: ISO 19005-1:2005/Corr 1:2007 (marzo 2007); ISO 19005-1:2005/Corr 2:2011 (dicembre 2011).

ISO 19005-2 (2011). «ISO 19005-2:2011; Document Management — Electronic document file format for long term preservation; Part 2: use of ISO 32000-1 (PDF/A-2)». URL http://www.iso.org/iso/catalogue_detail?csnumber=50655. ISO/TC 171/SC2.

ISO 19005-3 (2012). «ISO 19005-3:2012; Document Management — Electronic document file format for long term preservation; Part 3: use of ISO 32000-1 with support for embedded files (PDF/A-3)». URL http://www.iso.org/iso/catalogue_detail?csnumber=57229. Technical Committee ISO/TC 171/SC2.

ISO 24517 (2008). «ISO 24517-1:2008; Document management — Engineering document format using PDF — Part 1: Use of PDF 1.6 (PDF/E)». URL http://www.iso.org/iso/catalogue_detail?csnumber=42274. Technical Committee ISO/TC 171/SC2.

ISO 32000-1 (2008). «ISO 32000-1:2008; Document Management — Portable document format (PDF 1.7)». URL http://www.iso.org/iso/catalogue_detail?csnumber=51502. Technical Committee ISO/TC 171/SC2.

ISO 32000-2 (2014). «ISO 32000-2:20140220; Document management — Portable document format — Part 2: PDF2.0». Technical committee ISO/TC 171/SC2 (in bozza febbraio 2014).

- ISO PDFUA (2015). «PDFUA Technical Implementation Guide: Understanding ISO 14289-1 (PDFUA-1)». URL <http://www.aiim.org/Research-and-Publications/standards/committees/PDFUA/Technical-Implementation-Guide>. AIIM Global Community of Information Technologies.
- ISO15930-3 (2002). «ISO 15930-3:2002; Graphic technology — Prepress digital data exchange — Use of PDF — Part 3: Complete exchange suitable for colour-managed workflows (PDF/X-3)». URL http://www.iso.org/iso/catalogue_detail.htm?csnumber=34941. Technical Committee ISO/TC 130.
- ISO15930-4 (2003). «ISO 15930-4:2003; Graphic technology — Prepress digital data exchange — Use of PDF — Part 4: Complete exchange of CMYK and spot colour printing data using PDF 1.4 (PDF/X-1a)». URL http://www.iso.org/iso/catalogue_detail.htm?csnumber=39938. Technical Committee ISO/TC 130.
- ISO15930-8 (2010). «ISO 15930-8:2010; Graphic technology — Prepress digital data exchange — Use of PDF — Part 8: Partial exchange of printing data using PDF 1.6 (PDF/X-5)». URL http://www.iso.org/iso/catalogue_detail.htm?csnumber=55844. Technical Committee ISO/TC 130. Rivisto via Corrigendum: ISO 15930-8:2010/Cor:2011 (agosto 2011).
- ▷ C.V. Radhakrishnan
River Valley Technologies
Trivandrum, India 695014
cvr (at) river-valley dot org
<http://cvr.cc/>
 - ▷ Hàn Thế Thành
River Valley Technologies
Trivandrum, India 695014
thanh (at) river-valley dot org
 - ▷ Ross Moore
Mathematics Department
Macquarie University
Australia 2109
ross (at) mq dot edu dot au
<http://maths.mq.edu.au/~ross/>
 - ▷ Peter Selinger
Department of Mathematics and Statistics
Dalhousie University
Halifax, Nova Scotia B3H 4R2
Canada
selinger (at) mathstat dot dal dot ca
<http://www.mathstat.dal.ca/~selinger>

La chimica con L^AT_EX 2_ε

Un esame dei pacchetti esistenti e delle possibilità*

Clemens Niederberger

Sommario

Questo articolo espone un esame dei pacchetti più utili e conosciuti per scrivere di chimica con L^AT_EX 2_ε. Si basa su una serie di messaggi scritti dall'autore sul proprio blog (in tedesco): <http://texwelt.de/blog>.

Abstract

This article provides an overview of the most useful and popular packages for writing chemistry-related material with L^AT_EX 2_ε. It is based upon a series of blog posts written by the author on the (German-language) blog <http://texwelt.de/blog>.

1 Introduzione

Circa due anni fa, sul mio blog ho cominciato a scrivere una serie di messaggi sui pacchetti di L^AT_EX adatti alla chimica. I lettori interessati e che conoscono il tedesco possono trovarli all'indirizzo <http://texwelt.de/blog/tag.chemie>. Questo articolo è grosso modo una traduzione dal tedesco di quei contributi e segue la stessa struttura.

1. I fondamenti mediante l'uso del pacchetto `chemmacros` (paragrafo 2).
2. Le formule molecolari e le equazioni di equilibrio chimico (paragrafo 3).
3. Le formule di struttura con il pacchetto `chemfig` (paragrafo 4).
4. I dati di sicurezza e il GHS con il pacchetto `ghsystem` (paragrafo 5).
5. La numerazione dei composti con il pacchetto `chemnum` (paragrafo 6).
6. I pacchetti per applicazioni specialistiche (paragrafo 7).

Devo spiegare che sono l'autore della maggior parte dei pacchetti descritti. Tutti i pacchetti citati in questo articolo si trovano in CTAN (<http://ctan.org/pkg/<nome-del-pacchetto>>) e sono contenuti nelle due maggiori distribuzioni del sistema T_EX: MiK_TE_X e T_EX Live. Chiunque disponga di una distribuzione aggiornata può consultare i rispettivi manuali semplicemente eseguendo in un terminale il comando `texdoc<nome-del-pacchetto>`.

*Questo articolo è la traduzione in italiano di quanto ha pubblicato l'autore in inglese (NIEDERBERGER, 2015c). Eventuali errori sono da attribuire ai traduttori C. Beccari e T. Gordini. (N.d.T.)

2 I fondamenti con il pacchetto `chemmacros`

Il pacchetto `chemmacros` (scusatemi: l'ho scritto io) mette a disposizione una varietà di macro per diverse applicazioni in chimica: permette di scrivere i nomi dei composti secondo le regole della IUPAC (International Union of Pure and Applied Chemistry), consente di trattare cariche e numeri di ossidazione, di mostrare i dati spettroscopici e molto altro. Recentemente (fine agosto 2015) `chemmacros` è stato aggiornato alla versione 5.0, che contiene importanti innovazioni (NIEDERBERGER, 2015a). In questo articolo si fa riferimento alla versione 5.0.

A partire dalla versione 5.0 `chemmacros` presenta una struttura modulare. Alcuni moduli sono caricati per impostazione predefinita, mentre altri devono essere caricati dall'utente. Questo articolo carica un solo modulo aggiuntivo:

```
\usechemmodule{redox}
```

2.1 Nomi

I nomi IUPAC possono essere piuttosto lunghi e difficili da leggere. Inoltre, dividerli in righe diventa spesso problematico. Trattandoli come testo normale è improbabile che si ottengano buoni risultati tipografici:

```
(4-(4,4'-Bis(dimethylaminophenyl)benz%  
hydryliden)cyclohexa-2,5-dien-1-yliden)%  
dimethylammoniumchlorid
```

```
(4-(4,4'-  
Bis(dimethylaminophenyl)benzhydryliden)cyclohexa-  
2,5-dien-1-yliden)dimethylammoniumchlorid
```

`chemmacros` mette a disposizione la macro `\iupac` per facilitare sia l'immissione del nome, sia la sua composizione. Nell'argomento di questo comando, '-' produce un trattino che permette la cesura in fine di riga, in modo simile a quanto fa `\babelhyphen`. '|' inserisce un punto di cesura insieme a un piccolo spazio che può essere personalizzato.

```
\iupac{(4-(4,4'-Bis(di|methy|laminol%  
phenyl)benz|hydryliden)cyclo|hexa%  
-2,5-dien-1-yliden)di|methy|l%  
ammonium|chlorid}
```

TABELLA 1: Casi di base per mhchem e chemformula

mhchem		chemformula	
<code>\ce{H2O}</code>	H ₂ O	<code>\ch{H2O}</code>	H ₂ O
<code>\ce{Sb2O3}</code>	Sb ₂ O ₃	<code>\ch{Sb2O3}</code>	Sb ₂ O ₃
<code>\ce{H+}</code>	H ⁺	<code>\ch{H+}</code>	H ⁺
<code>\ce{[AgCl2]-}</code>	[AgCl ₂] ⁻	<code>\ch{[AgCl2]-}</code>	[AgCl ₂] ⁻
<code>\ce{^{227}_{90}Th+}</code>	²²⁷ ₉₀ Th ⁺	<code>\ch{^{227}_{90}Th+}</code>	²²⁷ ₉₀ Th ⁺
<code>\ce{SO4^{2-}}</code>	SO ₄ ²⁻	<code>\ch{SO4^{2-}}</code>	SO ₄ ²⁻
<code>\ce{KCr(SO4)2 * 12H2O}</code>	KCr(SO ₄) ₂ · 12 H ₂ O	<code>\ch{KCr(SO4)2 * 12 H2O}</code>	KCr(SO ₄) ₂ · 12 H ₂ O

(4-(4,4'-Bis(dimethylaminophenyl)benzhydryliden)cyclohexa-2,5-dien-1-yliden)dimethylammoniumchlorid

Se fosse solo per la cesura in fine di riga, una soluzione più semplice potrebbe essere quella di usare le scorciatoie di `babel`. Ma `\iupac` fa di più: dentro molte macro per comuni compiti di composizione, i nomi IUPAC sono già definiti, per esempio:

```
\iupac{\nitrogen-methyl|benz|amide} \\  
\iupac{\cip{2S,3S}-Wein|s|"aure} \\  
\iupac{\zusammen-2-Butene}
```

N-methylbenzamide
(2*S*,3*S*)-Weinsäure
(*Z*)-2-Butene

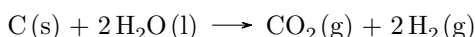
2.2 Descrittori di fase

`chemmacros` tenta di seguire il più possibile le raccomandazioni IUPAC. Per quanto riguarda i descrittori di fase, IUPAC richiede quanto segue:

I [...] simboli sono usati per rappresentare gli stati di aggregazione delle specie chimiche. Le lettere vengono messe accanto fra parentesi e sono composte in tondo (diritto) senza nessun punto finale. (COHAN *et al.*, 2008, p. 54)

Per i descrittori di fase più comuni, `chemmacros` mette a disposizione alcune macro pronte all'uso che seguono esattamente quelle raccomandazioni:

```
\ch{  
  C\sld{} + 2 H2O\lqd{}  
  ->  
  CO2\gas{} + 2 H2\gas  
}
```



Tuttavia, in quasi tutte le impostazioni `chemmacros` permette di personalizzare la composizione. La formula precedente con l'impostazione

```
\chemsetup{phases/pos=sub}
```

produce un altro modo molto frequente per indicare le fasi:

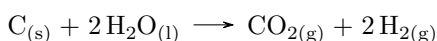


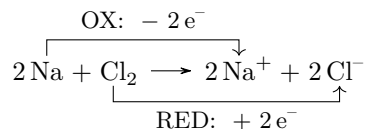
TABELLA 2: Le frecce di mhchem e chemformula

mhchem		chemformula	
<code>\ce{->}</code>	→	<code>\ch{->}</code>	→
<code>\ce{<-}</code>	←	<code>\ch{<-}</code>	←
<code>\ce{<->}</code>	↔	<code>\ch{<->}</code>	↔
		<code>\ch{<>}</code>	↔
<code>\ce{<=>}</code>	⇌	<code>\ch{<=>}</code>	⇌
<code>\ce{<=>}</code>	⇌	<code>\ch{<=>}</code>	⇌
<code>\ce{<=>}</code>	⇌	<code>\ch{<=>}</code>	⇌
		<code>\ch{<->}</code>	↔
		<code>\ch{<->}</code>	↔
		<code>\ch{<->}</code>	↔
<code>\ce{=}</code>	=	<code>\ch{=}</code>	=
		<code>\ch{<o>}</code>	↔

2.3 E molto di più

Siccome descrivere tutto quello che `chemmacros` può fare sarebbe piuttosto inutile (visto che è tutto descritto nella sua documentazione) e farebbe superare i limiti imposti alla lunghezza di questo articolo, darò solo un altro esempio ancora prima di passare all'argomento successivo. Si tratta di un'equazione di ossidoriduzione:

```
\ch{  
  2 "\OX{o1,Na}" + "\OX{r1,Cl}" {}2  
  ->  
  2 "\OX{o2,Na}" {}+ + 2 "\OX{r2,Cl}" {}-  
}  
\redox{o1,o2}[->]{\small OX: \ch{- 2 e-}}  
\redox{r1,r2}[->][-1]{\small RED: \ch{+ 2 e-}}
```



3 Formule molecolari ed equazioni di reazione – i pacchetti mhchem e chemformula

Qui bisogna citare due pacchetti: `mhchem` di Martin Hensel e il mio `chemformula`. Il primo è disponibile da più tempo e quindi è meglio conosciuto e più usato.

`chemformula` è nato come parte di `chemmacros` nel gennaio del 2012 ed è stato pubblicato come pacchetto indipendente nel luglio del 2013. È ancora strettamente collegato a `chemmacros` nel senso

che quest'ultimo lo usa piuttosto ampiamente. Perciò, coloro che usano `chemmacros` dovrebbero usare `chemformula` al posto di `mhchem` in modo da avere input e output coerenti.

Nonostante i due pacchetti siano molto simili per quanto riguarda ingresso e uscita, presentano importanti differenze in entrambi i casi.

Martin Hensel, l'autore di `mhchem`, una volta descrisse le differenze in questo modo (HENSEL, 2015):

La filosofia [di `chemformula`] è il *controllo in mano all'utente*. [...] la filosofia di `mhchem`, invece, è la *facilità d'uso*.

Entrambi i pacchetti offrono principalmente una sola macro per comporre formule e reazioni chimiche:

- `mhchem` ha `\ce{⟨formula⟩}`, e
- `chemformula` ha `\ch{⟨formula⟩}`.

La tabella 1 mostra un breve confronto dei casi d'uso più comuni.

Le differenze sono più visibili quando si compongono reazioni:

```
\ce{4 Na + 2 H2SO4 -> 2 Na2SO4 + 2 H2 ^} \\
\ch{4 Na + 2 H2SO4 -> 2 Na2SO4 + 2 H2 ^}
```

```
4 Na + 2 H2SO4 —> 2 Na2SO4 + 2 H2 ↑
4 Na + 2 H2SO4 —> 2 Na2SO4 + 2 H2 ↑
```

Si noteranno differenze nelle spaziature (personalizzabili con `chemformula`) e naturalmente il diverso aspetto delle frecce. La tabella 2 mostra i vari tipi di frecce disponibili con i due pacchetti.

Ci sarebbe molto altro da dire su entrambi i pacchetti, ma di nuovo devo accontentarmi di questa introduzione e rimandare alle rispettive documentazioni per i dettagli.

4 Formule di struttura – il pacchetto `chemfig`

Sia `mhchem` sia `chemformula` non forniscono alcun supporto per mostrare la complessità dei composti organici. Per comporre le formule di struttura, la strada più battuta è probabilmente quella di ricorrere a programmi esterni come `ChemDraw`, esportarne i risultati come immagini e includerle nel documento L^AT_EX. Tuttavia, esistono i metodi per creare formule e diagrammi di questo tipo all'interno di L^AT_EX. Per quanto ne so esistono cinque pacchetti per farlo. I lettori interessati possono riferirsi a RAKE (2012). In questo articolo ne prenderò in esame solo uno: `chemfig` di Christian Tellechea. E di nuovo sarà solo una breve introduzione, perché il manuale ammonta a di più di 80 pagine.

Prima di tutto: `chemfig` è un pacchetto generico (è l'unico di questo tipo fra quelli descritti in questo articolo); può essere usato con L^AT_EX, ConT_EXt e plain T_EX.

TABELLA 3: Tipi di legami di `chemfig`

Input	Output
<code>\chemfig{A-B}</code>	A — B
<code>\chemfig{A=B}</code>	A = B
<code>\chemfig{A~B}</code>	A ≡ B
<code>\chemfig{A>B}</code>	A ► B
<code>\chemfig{A<B}</code>	A ◄ B
<code>\chemfig{A>:B}</code>	A ... B
<code>\chemfig{A<:B}</code>	A ... B
<code>\chemfig{A> B}</code>	A ▷ B
<code>\chemfig{A< B}</code>	A ◁ B

```
\input chemfig.tex % plain TeX
\usepackage{chemfig} % LaTeX
\usemodule[chemfig] % ConTEXt
```

`chemfig` si serve di PGF (TikZ) per disegnare le formule. Il comando più importante è

```
\chemfig{⟨formula⟩}
```

Le regole di base sono semplici: le lettere sono atomi e i legami sono inseriti come `-`, `=`, eccetera. Gli atomi sono composti in modo matematico (più precisamente con `\printatom` che si espande in `\ensuremath{\mathrm{#1}}`), cosicché si possono inserire i pedici come si fa scrivendo la matematica:

```
\chemfig{H-CH_3}
```

H — CH₃

Esistono molti tipi di legame, come mostra la tabella 3. Il loro aspetto può essere personalizzato con un certo numero di parametri. Tutti i comandi dei legami hanno un argomento facoltativo che può ricevere un lista di 5 parametri, separati da una virgola, in quest'ordine:

1. l'angolo del legame;
2. un fattore di scala;
3. il numero dell'atomo di partenza;
4. il numero dell'atomo di arrivo;
5. le opzioni grafiche di TikZ per personalizzare il legame.

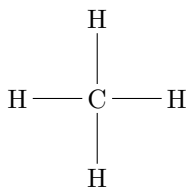
L'angolo può essere immesso in tre modi diversi:

1. `⟨intero⟩` — denota un angolo in multipli di 45°.
2. `:⟨reale⟩` — denota l'angolo (in gradi) del legame: il verso è antiorario se il valore è positivo e orario se è negativo.
3. `::⟨reale⟩` — denota un angolo (in gradi) antiorario se positivo, orario se negativo, relativo all'angolo specificato in precedenza.

Perciò, ciascuna delle seguenti linee di codice:

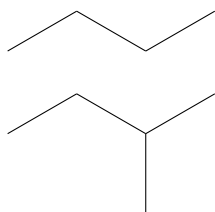
```
\chemfig{H-C(-[2]H)(-[6]H)-H}
\chemfig{H-C(-[:90]H)(-[:-90]H)-H}
\chemfig{H-C(-[:90]H)(-[:-90]H)-H}
```

produce lo stesso risultato:



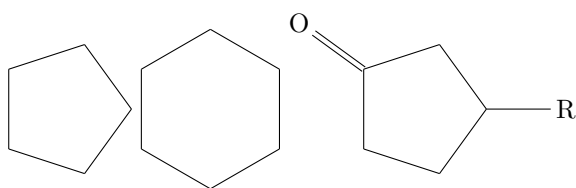
L'ultimo esempio mostra anche come si eseguono le diramazioni: basta mettere la specifica per la diramazione fra parentesi.

```
\chemfig{-[:30]-[::-60]-[::60]}
\par\bigskip
\chemfig{-[:30]-[::-60](-[::-60])-[::60]}
```



Anche creare gli anelli è piuttosto semplice:

```
\chemfig{*5(-----)}
\chemfig{*6(-----)}
\chemfig{*5(--(-R)--(=O)-)}
```



Personalmente ritengo che questa sintassi sia piuttosto intuitiva — certamente molto di più di quella di alcuni degli altri pacchetti menzionati in [RAKE \(2012\)](#) — e la può essere appresa molto in fretta.

Spero che gli esempi che ho mostrato diano una idea di base dell'uso di `chemfig`. Per concludere questo paragrafo, la figura 1 mostra un esempio di uno schema più complesso creato appositamente con `chemfig` per offrire una visione più ampia delle sue funzionalità. Il codice per l'esempio è riportato in [NIEDERBERGER \(2015b\)](#).

5 Informazioni sulla sicurezza — i pacchetti `rsphrase`, `hpstatement` e `ghsystem`

Ogni studente di chimica conosce le frasi di sicurezza armonizzate nel sistema *Globally Harmonized System and Labelling of Chemicals*, in breve GHS. Questo definisce le dichiarazioni dei tipi H (*hazard*, “pericolo”) e P (*precaution*, “attenzione”). Prima di questo standard c'erano i tipi R (*risk*, “rischio”) e S (*safety*, “sicurezza”). Le dichiarazioni di tipo R

e S sono disponibili mediante il pacchetto `rsphrase` di Martin Hensel¹:

```
The statement \rsnumber{R34}
is ‘\rsphrase{R34}’
```

The statement R 34 is ‘Provoca ustioni.’

L'uso di questo pacchetto è descritto nel manuale di `mhchem`.

Nello stesso manuale è descritto anche l'uso del pacchetto `hpstatement` dello stesso autore. Questo pacchetto fornisce il supporto per le dichiarazioni H e P come specificate dal GHS. È un peccato che il pacchetto contenga la specificazione

```
\RequirePackage{babel}
```

che obbliga l'utente a servirsi del pacchetto `babel` e a specificare una lingua come opzione².

```
\selectlanguage{english}%
The statement \hpnumber{H200}
is ‘‘\hpstatement{H200}’’
```

The statement H200 is “Unstable explosives.”

Esiste un altro pacchetto per le frasi GHS, il mio `ghsystem`, leggermente anteriore a `hpstatement`. Anch'esso fornisce un'interfaccia per gestire le frasi per numero.

```
The statement \ghs[hide]{h}{200}
is ‘‘\ghs*{h}{200}’’: \par
\ghs{h}{200}
```

The statement H200 is “Esplosivo instabile.”:
H200: Esplosivo instabile.

Questo pacchetto permette anche di inserire i pittogrammi GHS:

```
\ghspic{skull}
\ghspic{flame}
\ghspic{aqpol}
```



I pittogrammi sono disponibili nei formati JPG, EPS, PNG E PDF. Avrei in programma di fornire questi pittogrammi anche come codici `TikZ`.

1. Volutamente non si è tradotto in italiano il testo sorgente, in modo da evidenziare che il pacchetto, almeno per alcune lingue fra cui l'italiano, compone le frasi nella lingua predefinita o in una lingua definita mediante un'opzione. (*N.d.T.*)

2. Attualmente, con il pacchetto `hpstatement` le frasi sono disponibili solo in inglese e in tedesco. Poiché in questo articolo la lingua principale è l'italiano, si è dovuta specificare la lingua inglese, altrimenti il pacchetto produce un avviso d'errore, ben comprensibile perché non dispone della macro che contiene la frase in italiano. La documentazione di `hpstatement`, contenuta in quella di `mhchem`, invita gli utenti chimici a contribuire con le traduzioni nella propria lingua madre. (*N.d.T.*)

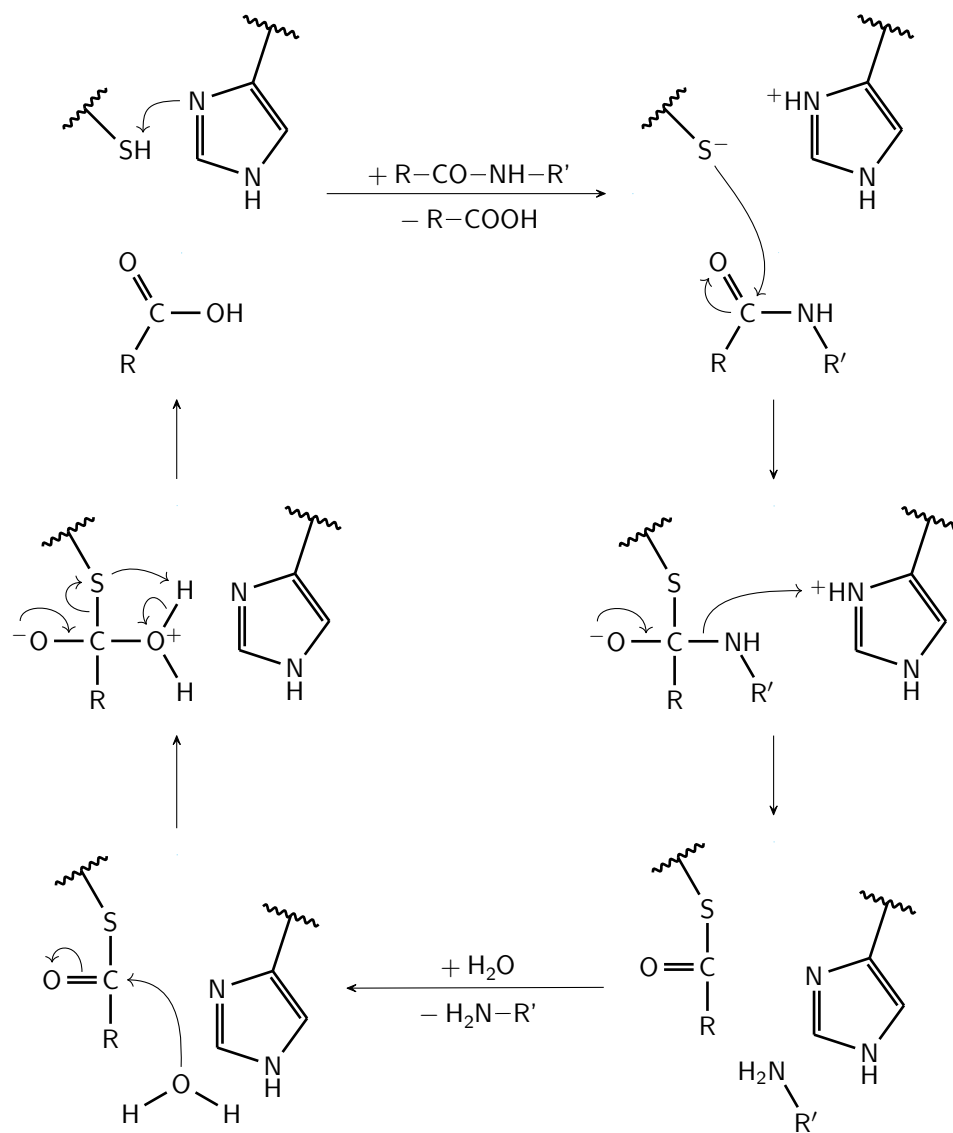


FIGURA 1: Uno schema complesso costruito con chemfig

6 La numerazione dei composti – il pacchetto chemnum

Per i chimici, riferirsi ai composti mediante dei numeri è un compito normale. Esistono alcuni pacchetti che permettono di farlo, ma quello più flessibile è il mio chemnum.

L'uso di base è semplice: basta aggiungere `\cmpd{<id>}` dove si vuole indicare il numero:

This is about `\ch{CH3-CH2-OH}` (`\cmpd{ethanol}`).
`\cmpd{ethanol}` is used\ldots

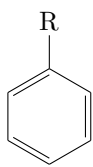
This is about CH₃-CH₂-OH (1). 1 is used...

La prima volta che si usa un ID `<id>` il comando crea un nuovo numero per il composto. Le occorrenze seguenti faranno riferimento a questo numero. Per l'uso nei titoli di sezione o nelle didascalie esiste una versione con un segno + facoltativo (`\cmpd+{<id>}`) che permette di riferirsi al composto senza creare il numero. C'è anche una versione asteriscata che invece crea il numero, ma non scrive nulla. Da questo punto di vista, la sintassi è la stessa del più anziano pacchetto chemcompounds di Stephan Schenck.

chemnum permette nativamente anche l'etichettatura subordinata: `\cmpd{<id>.<sub-id>}`.

Sia `<id>` sia `<sub-id>` possono essere liste di identificatori separati da virgole. Se `<sub-id>` è una lista, deve essere racchiuso tra graffe.

```
\setatomsep{2em}
\chemname
{\chemfig{*6(==--(-R)=)}}
{\cmpd{benzene.{H,Me,OH,NH2}}}
\begin{tabular}{lll}
& \ch{-R} & Name \\
\midrule
\cmpd[sub-only]{benzene.H}
& \ch{-H} & Benzene\\
\cmpd[sub-only]{benzene.Me}
& \ch{-CH3} & Toluene \\
\cmpd[sub-only]{benzene.OH}
& \ch{-OH} & Phenol \\
\cmpd[sub-only]{benzene.NH2}
& \ch{-NH2} & Phenylamine (Aniline)
\end{tabular}
```

		-R	Name
2a-d	a	-H	Benzene
	b	-CH ₃	Toluene
	c	-OH	Phenol
	d	-NH ₂	Phenylamine (Aniline)

Questo esempio mostra che per impostazione predefinita le liste di sottoetichette sono compresse. Si può disabilitare questo comportamento mediante un'opzione.

```
\cmpd{benzene.{H,Me,OH,NH2}}
\setchemnum{compress=false}
\cmpd{benzene.{H,Me,OH,NH2}}
```

2a-d 2a,b,c,d

Quando si dispone di una lista di etichette che contengono un `<id>` più di una volta ma con sotto etichette diverse, ciascuna citazione verrà stampata per conto proprio:

```
\cmpd{a,b.{A},c,b.{B,C},a}
```

3, 4a, 5 e 4b,c

Tuttavia, esiste un'opzione per unire quelle citazioni in una sola:

```
\setchemnum{merge=true}
\cmpd{a,b.{A},c,b.{B,C},a}
```

3, 4a,b,c e 5

Va notato ancora un dettaglio che non emerge dagli esempi precedenti, cioè che le liste sono elencate in ordine alfabetico:

```
\cmpd{a,b,c,b,a,ethanol,benzene} vs. \@
\cmpd{a}, \cmpd{b}, \cmpd{c}, \cmpd{b},
\cmpd{a}, \cmpd{ethanol} and \cmpd{benzene}
```

1, 2, 3, 4 e 5 vs. 3, 4, 5, 4, 3, 1 and 2

7 Pacchetti per applicazioni specialistiche

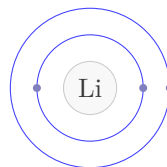
Il numero di pacchetti disponibili per gestire argomenti collegati alla chimica è piuttosto grande. Sia CTAN (<http://ctan.org/topic/chemistry>) sia il mio blog (<http://mychemistry.eu/known-packages>) ne mantengono un'elenco. Le due liste sono leggermente diverse, soprattutto perché nel mio blog includo anche argomenti di confine, per esempio pacchetti che possono essere legati più alla biologia che alla chimica. In questo paragrafo scelgo alcuni dei miei pacchetti e li descrivo molto brevemente:

- bohr,
- elements,
- endiagram,
- modigram, e
- chemgreek.

7.1 Il pacchetto bohr

Il pacchetto bohr è stato creato in risposta a una domanda inviata a <http://tex.stackexchange.com>, (ANDREAS, 2012). Esso permette di disegnare in modo semplice gli orbitali atomici secondo il modello di Bohr:

```
\bohr{3}{Li}
```



7.2 Il pacchetto elements

Il pacchetto `elements` è piuttosto recente (pubblicato nel giugno 2015). Dal suo sommario:

Questo pacchetto fornisce i mezzi per gestire le proprietà degli elementi chimici come numero atomico, simbolo dell'elemento, nome dell'elemento, distribuzione degli elettroni, numero degli isotopi. Queste proprietà sono definite per gli elementi fino al numero atomico 112.

L'esempio seguente dà una rapida idea delle sue potenzialità³:

```
\elementname{Cu} \\
\elementname{11} \\
\atomicnumber{U} \\
\elconf{Cl} \\
\savemainelementisotope\foo{C}\foo
```

Copper

Sodium

92

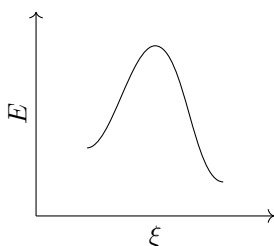
1s²2s²2p⁶3s²3p⁵

12

7.3 Il pacchetto endiagram

Il pacchetto `endiagram` permette di creare diagrammi di curva dell'energia potenziale.

```
\begin{endiagram}
\ENcurve{1,4,0}
\end{endiagram}
```

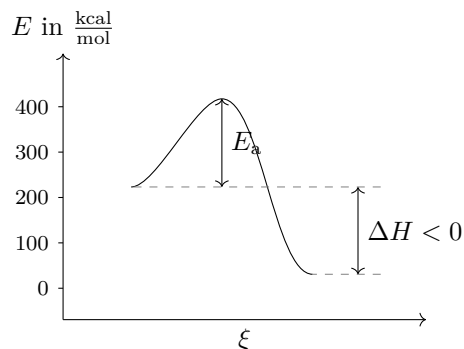


Un esempio un po' più complesso:

```
% preamble:
% uses 'siunitx' (loaded by 'endiagram')
\DeclareSIUnit{\calory}{cal}
\sisetup{per-mode = fraction}
% document:
\ENsetup{
y-label = above ,
energy-step = 100 ,
energy-unit = \kilo\calory\per\mole ,
energy-unit-separator = { in } ,
calculate = false ,
AddAxisLabel/font = \footnotesize
}
```

3. La stringa di controllo `\foo`, secondo la tradizione di D.E. Knuth, serve per designare una macro il cui nome è scelto dall'utente. (*N.d.T.*)

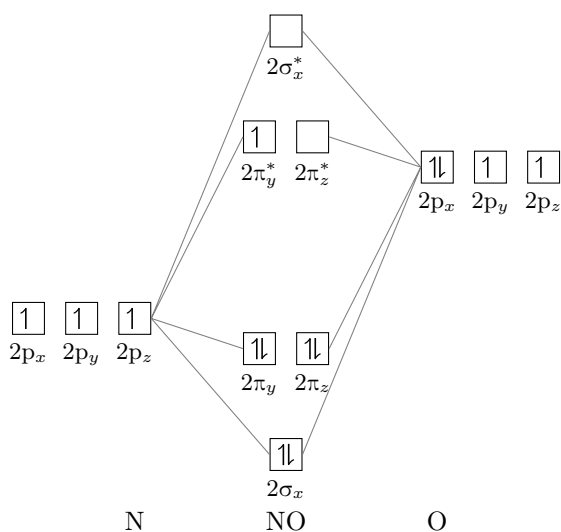
```
\begin{endiagram}[scale=1.2]
\ENcurve{2.232,4.174,0.308}
\AddAxisLabel*{0;1;2;3;4}
\ShowEa[label,connect={draw=none}]
\ShowGain[label]
\end{endiagram}
```



7.4 Il pacchetto modidiagram

Come il pacchetto `bohr`, il pacchetto `modidiagram` è stato creato in risposta a una domanda inviata a <http://tex.stackexchange.com>, (TERRETT, 2011). Esso offre un'interfaccia per disegnare diagrammi degli orbitali molecolari servendosi di `TikZ`.

```
\begin{Modiagram}[labels,names,style=square]
\atom[N]{left}{
2p = {0;up,up,up}
}
\atom[O]{right}{
2p = {2;pair,up,up}
}
\molecule[NO]{
2pMO = {1.8,.4;pair,pair,pair,up}
}
\end{Modiagram}
```



7.5 Il pacchetto chemgreek

Il pacchetto `chemgreek` è un pacchetto da usare più come supporto che in modo autonomo. Attualmente si servono delle sue funzionalità i pacchetti

chemmacros, chemnum e mhchem. Lo scopo di chemgreek è quello di fornire un'interfaccia unificata per le lettere greche diritte, indipendentemente da quale pacchetto per la matematica o per i font le fornisca e da come vi si acceda. La cosa è importante in chimica, poiché le lettere greche diritte sono usate in una moltitudine di circostanze:⁴

```
\iupac{\a-\D-glucopyranose}
```

α -D-glucopyranose

Il pacchetto chemgreek conosce un certo numero di pacchetti che forniscono queste lettere. È in grado di determinare se uno di quei pacchetti è già stato caricato e, se è possibile farlo in modo univoco, definisce le macro per accedere a ciascuna delle 24 lettere minuscole e delle corrispondenti maiuscole. Se la cosa non è possibile, si rifà a una configurazione predefinita e gli utenti devono provvedere da soli ad eseguire una scelta. Il pacchetto chemgreek ha una comoda interfaccia allo scopo:

```
\selectchemgreekmapping{\langle mapping \rangle}
```

Ora altri pacchetti possono usare quelle macro per definirne altre per conto proprio.

Si noti che il pacchetto riconosce i $\langle mapping \rangle$ con i nomi dei pacchetti che forniscono quelle lettere, per esempio newtx per usare i font greci diritti della collezione newtxmath; textalpha per usare (anche) in matematica i font greci testuali diritti predefiniti per la lingua greca, eccetera. È opportuno riferirsi alla documentazione del pacchetto chemgreek.⁵

Se un documento richiede un pacchetto che contiene lettere greche diritte di cui chemgreek non sa nulla, gli utenti possono definirsi un nuovo $\langle mapping \rangle$ da soli e (per esempio) attivarlo con l'interfaccia di chemmacros e poi usare i comandi di nomenclatura di chemmacros per usare il nuovo mapping.

8 Conclusioni

In questo articolo si sono descritti brevemente i pacchetti chemmacros, mhchem, chemformula, chemfig,

4. Si noti che nella formula seguente la lettera 'D', codificata con \D, non è una 'D' maiuscola, ma è maiuscoletta, perché questo simbolo per IUPAC rappresenta il nome *dexter*. (N.d.T.)

5. Questo capoverso è stato aggiunto dai traduttori, che ritengono che la parola *mapping* non sia abbastanza chiara per i lettori italiani in questo contesto. (N.d.T.)

rsphrase, hpstatement, endiagram, modigram e chemgreek. Si è dimostrato che oggi esiste un grande supporto per i diversi compiti di composizione richiesti in chimica e si è vista una serie di possibilità d'uso dei pacchetti citati.

Riferimenti bibliografici

ANDREAS (2012). «Draw Bohr atomic model with electron shells in T_EX». URL <http://tex.stackexchange.com/questions/73410/>.

COHAN, E. R., CVITAŠ, T., FREY, J. G., HOLMSTRÖM, B., KUCHITSU, K., MARQUARDT, R., MILLS, I., PAVESE, F., QUACK, M., STOHNER, J., STRAUSS, H. L., TAKAMI, M. e THOR, A. J. (2008). *Quantities, Symbols, and Units in Physical Chemistry*. IUPAC & RSC Publishing, Cambridge, 3^a edizione.

HENSEL, M. (2015). «On mhchem vs. chemformula». URL http://tex.stackexchange.com/q/237946#comment565361_238214.

NIEDERBERGER, C. (2015a). «A new chemmacros – but how?». URL <http://www.mychemistry.eu/2015/07/a-new-chemmacros-but-how/>.

— (2015b). «A seemingly complex example for the usage of chemfig». URL <https://www.overleaf.com/read/nhbyvqvrqffj>.

— (2015c). «Chemistry in L^AT_EX 2_ε – an overview of existing packages and possibilities». *TUGboat*, **36** (3), pp. 227–234.

RAKE (2012). «Can you make chemical structure diagrams in L^AT_EX?». URL <http://tex.stackexchange.com/questions/52722>.

TERRETT, R. (2011). «Molecular orbital diagrams in L^AT_EX». URL <http://tex.stackexchange.com/questions/13863>.

▷ Clemens Niederberger
Herrenberg DE
contact (at) mychemistry (dot)
eu

Latin challenges

Luigi Scarso

Abstract

A history of Latin since 7000 BC until today is presented, along with some code to typeset Old Italic documents. The paper shows how the application of features written on-the-fly can solve some problems of representation and content rendering.

Sommario

L'articolo offre una concisa storia della lingua latina dal 7000 a.C. ai giorni nostri. Viene discusso un esempio di documento in Old Italic che evidenzia la relazione tra presentazione e contenuto e come l'applicazione di caratteristiche appositamente definite possano risolvere alcuni problemi di tale relazione.

1 Introduction

Patterns, a fundamental part of a typesetting engine, are shipped with the ConT_EXt distribution and kept in sync with the T_EXLive, and each new pattern—even an update of an existing one—is first checked by experts to verify the correctness and the backward compatibility.

It is not surprising then that the new version of the Latin language patterns by C. Beccari created a bit of bustle, especially because there is a new set of patterns for classical Latin—that ended in ConT_EXt as a new tag `ala`, while the previous tag `la` is now for medieval and modern Latin—which triggered some questions about Latin, its history and its “vitality”. These questions raised interesting connections with (or better, challenges for) the contemporary T_EX typography.

The next section outlines the history of Latin, highlighting some aspects that will be developed later.

2 A short history of Latin

Proto-Indo-European (7000–3000 BC)

A hypothesis widely accepted by linguists (the *Kurgan hypothesis*, (WIKIPEDIA, 2015A)) says that a lot of the current and old languages of a region ranging from Europe to the Indian subcontinent, across Iran and Afghanistan, have a common ancestor: a language (designed as *proto-indo-european* or PIE) spoken around 7000 BC in the Pontic-Caspian steppe. The hypothesis descri-

bes 3 waves of expansion from the origin (*Urheimat*). These waves are marked out by 3 stages of evolution of the language itself: early PIE (4500–4000 BC), middle PIE (3500–3000 BC) and late PIE (3000–2500 BC). After the last wave, the spoken language disappeared as a single entity. Due to these waves, the language reached modern Romania, Bulgaria, eastern Hungary and Georgia.

Proto-Italic(?) (3000–1000 BC)

During the III millennium BC, waves from Central Europe propagate to South-West and South-East, very likely reaching modern Italy from North-East Alps around the 2nd millennium BC. Some scholars *reconstructed* a sub-family of late PIE called *Proto-Italic* (BAKKUM, 2009) as the common ancestor of some important subsequent languages of the Peninsula (e.g., Sabellic and Latino-Faliscan) but, since there are no textual sources, Proto-Italic is currently not tied to a specific Italic region and could have originated outside the Peninsula. Archaeological findings also unveiled another language, spoken in a region corresponding to the modern Veneto region, (PETRINI, 2014) which could be a direct descendant of late PIE (some scholars talk of *Italic-Venetic*) and a non-PIE language (*Etruscan*) in modern Tuscany, western Umbria and northern Latium.

Italic (1000–550 BC)

Around 7th century BC there were at least two languages descending from Proto-Italic—Sabellic and Latino-Faliscan (SLOCUM, 2014). Other classifications see Venetic as another Italic descendant, but this hypothesis is not yet widely accepted. Two other languages, Aequeian and Vestinian, are also descendants of *Italic*, nevertheless there are too few findings to state this with certainty. This period marks the transition between prehistoric and documented phase, and it is the period of the Rome foundation, supposedly around 754/753 BC. Fig. 1 depicts the situation around 600 BC, when Umbrian and Oscan were part of the Sabellic group: it is evident the relevance of Greek and Etruscan and the presence of another late PIE descendant (*Messapic*) unrelated to all of the other languages.

The origins of Latin are influenced by several languages, and this is evident in the ancient shrine, known as *Lapis Niger* (Black Stone), where there is one of the oldest Latin inscriptions (fig. 2, 575–550 BC).

The lettering is very close to Greek letters and the inscription is written boustrophedon; it is cur-



FIGURE 1: Italic, 600 BC.

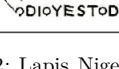
Lapis Niger (inscripción de sus cuatro caras)			
a			5
b			15
c			
d			

FIGURE 2: Lapis Niger, inscription, 575–550 BC.

rently the oldest Italic inscription that exhibits the word *sakros*, i.e., *sacred*. Very likely the Roman alphabet was a modification of the Etruscan one, which in turn could be derived from the Greek alphabet of the Comae colony (in the modern province of Naples). This is an Etruscan alphabet (*Marsiliana tablet*, 700 BC), with the transliteration in modern Greek:

ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΨΣΤΥΧΦΥ
ΑΒΓΔΕΖΗΘΙΚΛΜΝ*ΟΡ*ΠΡΣΤΥΞΦΧ
Asterisks are just placeholders for missing Greek characters; the Etruscan ones correspond to the English sound 'sh'. The 'H' Etruscan glyph corresponds to the English sound of 'h', not to the long Greek vowel 'H'.

That period saw the rise of Rome, that ended with the domination of Latin over all of the other languages of the Peninsula. Following (MONDIN, 2012) the periods are the following ones.

Pre-Literary Latin (550–241 BC)

The textual documentation is limited but shows the alphabet gradually moving from Etruscan to Roman (SAMPSON, 1985): this is the alphabet used in Etruria around 400 BC

ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΨΣΤΥΥ

and this is the alphabet used by Romans during the same time:

ΑΒΓΔΕΖΗΘΙΚΛΜΝΟΠΡΨΣΤΥΧ

For example, the Ficoroni cista (fig. 3) alrea-

dy shows a clear Roman alphabet (DANIELS AND BRIGHT, 1996), even though the "A" is still Italic. The text says: *Novios Plautios med Romai fecid. Dindia Macolnia fileai dedit* and it's one of the first inscriptions with the word "Rome".

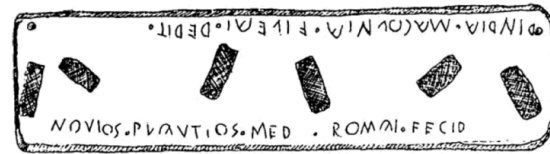


FIGURE 3: Ficoroni cista, latin inscription (350 / 325 BC).

Archaic Latin (241–79 BC)

The textual documents became more relevant and the alphabet started to get stability: in the 3rd century BC it is

ΑΒΓΔΕΖΗΘΙΚΛΜΝΟΠΡΨΣΤΥΧ

but already around 1st century BC the letters (called *Square Capitals*) were:

ΑΒΓΔΕΖΗΘΙΚΛΜΝΟΠΡΨΣΤΥΧΥΖ

Classical Latin (79 BC -14 AC)

This is the well-known period of Marcus Tullius Cicero, Gaius Julius Caesar, Gaius Sallustius Crispus, Gaius Valerius Catullus (Republican period) and Publius Vergilius Maro (Virgil), Quintus Horatius Flaccus (Horace), Publius Ovidius Naso (Ovid), Titus Livius (Augustan period). The alphabet evolved considerably: in addition to a formal one used for official documents and inscriptions there is an informal one, the *Rustic Capitals* (fig. 4, (MARCOS, 2014)):



FIGURE 4: Rustic Capitals: De Bello Actico, 31 BC - 79 AD.

In this period is also proved the existence of an Old Roman Cursive alphabet (fig. 6, (MARCOS, 2014)) where the letters were still uppercase but with some glyphs very similar to the lowercase counterpart (e.g. the "D" in fig. 7).

Post-classical Latin (14 AD - 193 AD)

This is the period of the Pax Romana and, thanks to Trajan, of the greatest military expansion. The Square Capital used for his inscription (fig. 5) is considered a canonical form. Latin was the official language of the Roman empire and it was increasingly used for administrative and commercial tasks. For this reasons, it started to spread

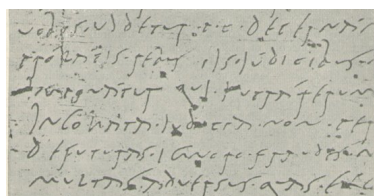


FIGURE 6: Papyrus
Claudius, 41 - 54 AD.

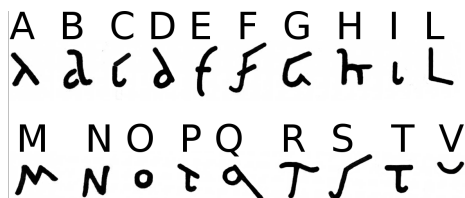


FIGURE 7: Old Roman Cursive.

outside the élite. As a consequence, the separation between formal/written literary Latin and informal spoken Latin was already observable at the end of this period.



FIGURE 7: Old Roman Cursive.

Late Latin (193 AC - 476 AD)

The distance between formal and “pure” Latin and *low* Latin increased. A new cursive (*New Roman Cursive* (MARCOS, 2014), fig. 8) appeared, and it was consistently used as minuscule also in ordinary documents. The end of the 2nd century saw the rise of the Christian religion, initially spread among the poor but later (edit of Milan, 313 AD) officially admitted and even promoted as a way to keep the unity of the empire; *catholicism* comes from the Greek term καθολικισμός (katholikismos, i.e., “universal”) and it has been used for the first time in The Epistles of St. Ignatius (108 AD, in Greek) to assert the role of a unique Church for the *entire* mankind in contrast with several different local Churches (Smyrna in this case), but the emperors pushed instead the use of Catholicism as universal, i.e., mandatory, for all the empire.

This had two opposite consequences: on one side, it enforced the role of the spoken Latin that will later die as a living language originating the Continental Romance; on the other side, the adoption of Classical Latin in its new role of supporting the emperor saved the language from extinction. Greek language had a similar evolution: initially most part of the Christians documents (included the Old and New Testament) were written

in Greek, and in the first seven councils (323–787 AD) Greek remained the reference language for the dogmatic definitions.

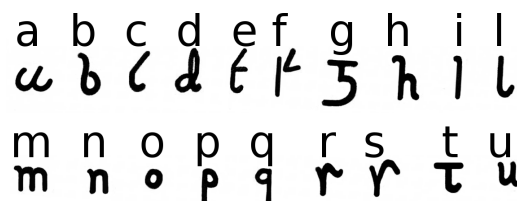


FIGURE 8: New Roman Cursive.

The importance of the period is attested by another alphabet that started to gain consensus especially among Christian authors: the *Uncial*, probably derived both from New Roman Cursive and Greek (the Codex Vaticanus, 325–350 AD is a Greek Bible) but increasingly used also for Latin, as in the Codex Victor (STEFFENS, 2015) in fig. 9.

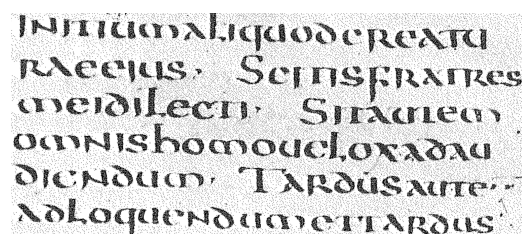


FIGURE 9: Codex Victor, Uncial, 546 AD.

Uncial (from Latin *uncia*, “one twelfth”) is a kind of transitional font: it’s a majuscule (i.e. upper case) alphabet with some minuscule letters, such as “p”, “q” and “h”. It also acts as a “bridge” between the paganism to reject and evangelism to pursue: the New Roman Cursive is too informal for the Christian literature and the Rustic Capital and Square Capital are both too tied to the old world, while the Uncial is more elegant and blends well initially with Greek and with Latin alphabet later. While this period ended with the Fall of Rome, it did not see the fall of Latin as a language because it kept being used as *the* language of literature and religion.

Medieval Latin (476 AD–first half 14th century)

This was the last period of Latin as a living language. The crisis of the Western Empire left the Christian Church as the strongest authority, but its initial vocation to universality was toward all the men and not military based, even though the 7th century saw the birth of the Papal State as a political entity. As a consequence of the absence of a strong political authority, Latin of that time was influenced by spoken Latin and by languages like Germanic and Gallic. The invasion of the Iberian Peninsula in 711 AD by Arabs, as well as their occupation of Sicily between 831 AD and 1072, was another source of influence because it broke

for the first time the Christian Church universal role. It is well known that the terms algorithm (al-Khwārizmī) and algebra (al-ğabr) were caught from the author Muhammad ibn Mūsā al-Khwārizmī and his book *Kitāb al-muhtaṣar fī ḥisāb al-ğabr wa-l-muqābala* (830 AD).

To set a date, the Council of Tours in 813 AD decided that priests should preach sermons in *rusticam romanam linguam* (rustic romance language) or Theodiscam (German) (WIKIPEDIA, 2016A) and one of the first known official documents, the Oaths of Strasbourg (842 AD), is written in three different languages: Medieval Latin, Old French and Old High German. The Old French passages are generally considered to be the earliest texts in a language that is distinctly French (WIKIPEDIA, 2015B). The 9th century can be hence considered as the beginning of the end of Latin as a living language.

Renaissance Latin (14th–15th century)

During the European Renaissance, a strong cultural movement (*humanism*) enforced the return to the origins (*ad fontes*) of Classical Latin. As a result, from that time on *Renaissance Latin* (or humanist Latin) became the standard. The Papal State is a defined political and geographical entity and Latin is the official language.

Neo-Latin (15th–20th century)

Humanist Latin spread in almost all fields of education (in 1687 Isaac Newton published the first edition of *Philosophiæ Naturalis Principia Mathematica* in humanistic Latin) and also diplomacy made an extensive use of Latin as international language. Catholic Roman Church kept using Latin but Martin Luther, between 1522 and 1534, published the *Luther Bible* in German, marking the beginning of the end of Latin as the only official language in Christianity. From 1700 to 1900, Latin gradually disappeared in Europe from scientific literature, laws and diplomacy (the last international treaty to be written in Latin is the Treaty of Vienna, written in 1738). The Papal State as an entity ceased to exist in October 1870, after a short conflict with the kingdom of Italy.

It's worth noting that up to the end of Middle Ages there are about 600 written texts, but afterwards their number rises to 18 000. The Gutenberg printing revolution played a fundamental role, marking the start of traditional typography.

Contemporary Latin (20th century–today)

By the mid of the 20th century, Latin was gradually dropped out of the core instruction. Humanistic Latin used by Catholic Roman Church slowly evolved in *Ecclesiastical Latin*, practically a minor variation of Classical Latin. The Lateran Treaty with Italy in 1929 created the State of the Vatican City in Rome, with Italian as the official language.

The State hosts the Holy See, i.e., the episcopal see of the Pope, representing his authority over the Catholic Church. The official language of the Holy See is Ecclesiastical Latin. Latin is used in theological works, liturgical rites, dogmatic proclamations and documents of Holy See. The Second Vatican Council (1962–1965) made Latin alternative, no longer the exclusive language of liturgies. Despite that, the definition of Latin as a *dead language* appears incorrect, because it's still in use for administrative tasks as shown in fig. 10.



FIGURE 10: ATM in Vatican, 2008.

In the final part of 20th century, the development of computer technologies marked the transitions from traditional to digital typography and, as a consequence, a renewed interest for the Latin documents as a source of best characters (as the Trajan typeface from Carol Twombly for Adobe, 1989). Unicode Consortium in 1991 published the first volume of UNICODE, “a computing industry standard for the consistent encoding, representation and handling of text expressed in most of the world’s writing systems” (WIKIPEDIA, 2016B). In this book, thanks to the large amount of documents, Latin glyphs are adequately represented. For Old Italic, on the contrary, the situation is quite different.

3 Some challenges for contemporary typography

Contemporary typography is almost completely digital typography and as such it cannot be separated from the type of digital support. In this paper, the support is the PDF file format, subject to two constraints: 1) the font is a “good” (i.e., well designed and with large number of glyphs) OPENTYPE and 2) the content derived from the PDF has to be UNICODE-compliant (i.e., “copy-and-paste” operations must preserve characters). These are quite reasonable requests for modern languages, but what about the Lapis Niger shown in fig. 2?

UNICODE supports Old Italic from 2000, so the first step is to find a good OPENTYPE font; AE-GEAN (DOUROS, 2015) meets the requirements. Installing the font in ConTeXt is a matter of co-

pying the files into a suitable location such as, for example, `tex/texmf-fonts` and issue a `mtxrun --generate` to update the cache. Then

```
\definefontsynonym[aegeanhint][file:Aegean.ttf]
\definefontsynonym[ArchaicLatin][aegeanhint]
```

after that, the font is available. Just make ConT_EXt use it with

```
{\definedfont[ArchaicLatin at 12pt] A B }
```

(ConT_EXt widely uses layered declarations to decouple the real file name and the abstract font name). The text of the Lapis Niger is written boustrophedon, as the red arrows show. It's then natural to define a similar set of macros:

```
\def\←#1{\righttoleft #1}
\def\↔#1{\bgroup% right to left & mirror
\setbox\nextbox\hbox{#1}%
\hbox to\hsize{\hss\vbox{\hsize=\wd\nextbox%
\righttoleft\mirror{\unhbox\nextbox}}}%
\egroup}
\def\↔#1{\hbox to 0.5\textwidth{\hss{\lefttoright #1}}}
\def\↖#1{\rotate[rotation=180,location=high]{#1}}
\def\↗#1{\mirror{#1}}
```

so that typesetting is straight:

```
\definedfont[ArchaicLatin at 9.8pt]
\←{Q Y O I B O I}
\↔{A K P O S E S}
\↔{E D \↔ O P M}
\↔{I A I A S}
\↔{P E C E I I C}
\↔{E Y A M}
\↔{Q Y O \↔ S P I}
\↔{\↔{\↔} : K \↔ \↔{\↔} \↔ \↔ T O}
\↔{\↔{\↔{P}} E \↔{\↔} : B A I}
\↔{O I O D : I Q Y X M E M}
\←{T A : K A \↔ I A : \↔ O T A Y}
\↔{M : I T E : P I A}
\↔{M : Q Y O I B A}
\↔{Y E L O D : M E Q Y}
\↔{O D I O Y E S T O D}
```

(as a side note, the previous macros implement a simple notation that helps discovering which characters are transformed).

The result is in fig. 13 and it can be verified that the representation matches the source inscription. The next step is to verify if the content matches the representation, but unfortunately this is not always true: for example, in the fifth row from the end (*ta kapia dotav*), the text copied is

Y A T O D : A I D A K : A T

where $\mathcal{D}$ and $\mathcal{D}$ are not mirrored. This is a limit of UNICODE: there is no modifier for this kind of transformations and the only solution is to add a codepoint to the standard as, for example, (EVERSON, 2016) proposed for a similar problem.

Following this example, a question arises: what does it happen if the image of a glyph is dubious and there are not alternative fonts? One answer is to create a font ad-hoc, i.e., with few glyphs, but this is not always the right answer. The design of an OPENTYPE font is a difficult and time-consuming task and a low number of glyphs usually means low quality. The alternative METAFONT produces Type 3 fonts, which is excluded because it is not OPENTYPE, and bitmaps in OPENTYPE are not widely used. The solution this paper proposes is a mix: define a new font feature on-the-fly to substitute just the needed glyphs with a bitmap image previously generated with METAFONT (it is basically an extension of the virtual font mechanism already used in T_EX).

The code starts registering the features `mfgl` and `mfgln` (names are arbitrary) for bitmap glyphs scaled / not scaled to the current fontsize:

```
\startluacode
fonts.handlers.otf.features.register {
    name      = "mfgl",
    description = "Add some metafont bitmap glyphs,
scaled to the current size.",
    manipulators = {
        base = addmfgl,
        node = addmfgl,
    }
}
fonts.handlers.otf.features.register {
    name      = "mfgln",
    description = "add some metafont bitmap glyphs, not scaled",
    manipulators = {
        base = addmfgl,
        node = addmfgl,
    }
}
\stopluacode
```

Both the features inherit from the `default` feature and add their own manipulators:

```
\definefontfeature[mfgl][default][mfgl=true]
\definefontfeature[mfgln][default][mfgln=true]
```

The features can be associated to a font by means of `\definefontsynonym` or directly with `\definedfont`:

```
{\definedfont[Aegean.ttf*mfgl at 12pt] this font has the feature
mfgl}
{\definedfont[Aegean.ttf*mfgln at 12pt]this font has the feature
mfgln}
```

Fig. 14 shows the LUA code implementing the features; it is not hard to explain the code: it takes the bitmap images already stored in the table `MFGlyphimage`, which is indexed with UNICODE codes, and replaces each glyph of the original font

with the related bitmap image. This is possible because the PDF has a glyph effect that does not draw the character and LuaT_EX can inject this PDF sequence as a part of a more complex command (please refer to the section *Virtual fonts* of the LuaT_EX manual for a detailed explanation). The net effect is a replacement of the glyphs keeping the ToUnicode mapping untouched, i.e., preserving the content.

The final step is to show how to produce a bitmap with METAFONT. First, it is important to recall that a pk font file can be translated into a hexadecimal bitmap with the program `pk2bm`, obtaining an ASCII file as the following (for the sake of simplicity most of the rows are omitted):

```
character : 65 (A)
  height : 351
  width : 203
  xoff : -15
  yoff : 327

00000000000000000000000000000000ff80000000000000000000
000000000000000000000000000000003ffff000000000000000
00000000000000000000000000000000ffff0000000000000000
000000000000000000000000000000003ffff0000000000000000
000000000000000000000000000000007ffff0000000000000000
00000000000000000000000000000000fffff8000000000000000
000000000000000000000000000000001fffffc000000000000000
000000000000000000000000000000001fffffe000000000000000
:
0fffff00000000000000000000000000000000000000000000000
07ffffe00000000000000000000000000000000000000000000000
01ffffc00000000000000000000000000000000000000000000000
00ffff000000000000000000000000000000000000000000000000
003ffc000000000000000000000000000000000000000000000000
```

The first step is hence to store these files into a LUA table with the right UNICODE index. Next, every hexadecimal glyph is translated into a PDF image object that uses the stencil masking, so that black pixels are transferred into the page, while the white ones (the background) are left unchanged. These images are then stored into the `MFglyphimage` global table (fig. 15). The result is shown in fig. 11: The first step is hence to store these files into a LUA table with the right UNICODE index. Next, every hexadecimal glyph.

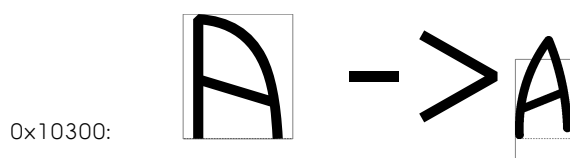


FIGURE 11: Original (left) and bitmap (right) U+10300 glyph.

The first step is hence to store these files into a LUA table with the right UNICODE index. Next,

every hexadecimal glyph.

Copy-and-paste is also preserved: `0x10300:A ->A` and the fonts are

```
$> pdffonts sample.pdf
name                                     type
-----
XNQYLW+Aegean                          CID TrueType
```

so there is no Type 3 font in the PDF.

Instead of bitmap images it is possible to use MetaPost code, but some workflows use METAFONT as an intermediate step in the production of OPEN-TYPE. In this case, the bitmap glyph eventually can be later replaced with a vectorial version.

The last challenge is the Ecclesiastical Latin character *œ*, which is absent from the UNICODE repertoire. It's quite a rare character, in use at least from 1894 (CHURCH, 1894) and the correct way to obtain it is by combining U+0153 and U+0301 (which is the combining acute accent). This preserves the content but the representation can be ugly as is in the case of Latin Modern: the accent is not correctly positioned. This is really a font problem, but it is possible to define an on-the-fly feature as seen before:

```
\startluacode
  fonts.handlers.otf.addfeature {
    name = "ptest",
    type = "pair",
    data = {
      [0x0153] = { [0x0301] = { false, { -75, -40,
0, 0 } } },
    }
  }
\stopluacode
\definefontfeature[default][default][ptest=yes]
```

The word *onomatopœia* in Linux Libertine at 12pt is shown in fig. 12: of course the patch is only valid for a specific font at a specific size, as viewable in the figure caption for Latin Modern.

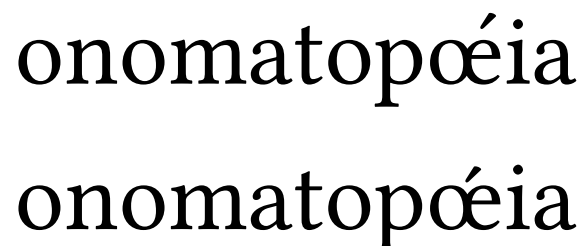


FIGURE 12: *onomatopœia* patched for the font Linux Libertine, before (top) and after (bottom).

This can appear a negligible difference, but (PER

L'ECUMENISMO E IL DIALOGO, 1999) explains that, during Middle Ages, *pænitentia* (penance) was transcribed, due an erroneous interpretation of *æ*, as *pænitentia* (the italic font well shows how it is possible to make such kind of mistakes), where *pœna* is punishment. This had an enormous influence on the Sacrament of Confession and only at the end of 20th century, also supported by these “typographical” considerations, the Sacrament gradually changed its name in Sacrament of Reconciliation, more aligned to the original meaning.

4 Conclusion

The history of Latin shows that even a very old language reserves surprises when meeting modern technologies. For Old Italic, it is clear that UNICODE does a great job, but it is not yet complete for the current needs of a digital document. Such needs require every domain to have its characters completely specified so that a simple search will not fail. For Neo-Latin the situation is good, because even in case of *æ* it's easy in ConTeXt to adjust the representation if it is suboptimal for a given font.

The author wishes to thank C. VINCOLETTA for his work with METAFONT on the font LapisNiger, used in section 3.

Bibliography

Links checked on 2016.02.13.

BAKKUM, G.-C.-L.-M. (2009). *The Latin Dialect of the Ager Faliscus: 150 Years of Scholarship*. (no. v. 1). Amsterdam University Press.

CHURCH, C. (1894). *Missale romanum: en decreto sacrosancti Concilii Tridentini restitutum*. URL: <https://archive.org/details/missaleromanume01churgoog>.

DANIELS, P.-T. & BRIGHT, W. (1996). *The World's Writing Systems*. Oxford University Press.

DOUROS, G. (2015). *Unicode Fonts for Ancient Scripts*. URL: <http://users.teilar.gr/~g1951d/>.

EVERSON, M. (2016). *Proposal for the addition of three Old Italic characters*. URL: <http://www.unicode.org/L2/L2015/15181r-n4669r-olditalic.pdf>.

MARCOS, J.-J. (2014). *Fonts for Latin paleography*. URL: http://guindo.pntic.mec.es/jmag0042/LATIN_PALEOGRAPHY.pdf.

MONDIN, L. (2012). *Introduzione allo studio del latino — dispensa dei corsi 2009-2010*. URL: [http://lettere2.unive.it/lmondin/Introduzione allo studio del latino](http://lettere2.unive.it/lmondin/Introduzione%20allo%20studio%20del%20latino).

PETRINI, S. (2014). *Storia, vita, costumi, religio-*

sità dei Veneti antichi. URL: <http://www.venetoimage.com/svc.htm>.

PER L'ECUMENISMO E IL DIALOGO, C.E.I.S. (1999). *Lettera di collegamento nr. 53*. URL: https://www.chiesacattolica.it/cci_new/documenti_cei/2001-02/13-25/lettera35.pdf.

SAMPSON, G. (1985). *Writing Systems: A Linguistic Introduction*. Stanford University Press.

SLOCUM, J. (2014). *Indo-European Languages — Evolution and Locale Maps*. URL: <http://www.utexas.edu/cola/centers/lrc/general/IE.html>.

STEFFENS, F. (2015). *Paléographie latine*. URL: http://www.icar.beniculturali.it/biblio/_view_volume.asp?ID_VOLUME=51.

WIKIPEDIA (2015A). *Kurgan hypothesis*. URL: https://en.wikipedia.org/wiki/Kurgan_hypothesis.

WIKIPEDIA (2015B). *Oaths of Strasbourg*. URL: https://en.wikipedia.org/wiki/Oaths_of_Strasbourg.

WIKIPEDIA (2016A). *Council of Tours*. URL: https://en.wikipedia.org/wiki/Council_of_Tours.

WIKIPEDIA (2016B). *Unicode*. URL: <https://en.wikipedia.org/wiki/Unicode>.

Source of figures

- fig. 1: Di Dbachmann, CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=333677>
- fig. 2: By <http://imperiormano.com/> - <http://imperiormano.com/imagenes/galeria/1174089912.jpg>, CC BY 3.0, <https://commons.wikimedia.org/w/index.php?curid=7710498>
- fig. 6: Original uploader was Avitus at la.wikipedia - Originally from la.wikipedia; description page is/was here., Public Domain, <https://commons.wikimedia.org/w/index.php?curid=2504913>
- fig. 7: By Osado - Own work, Public Domain, <https://commons.wikimedia.org/w/index.php?curid=6505958>
- fig. 5: By Photo of the inscription on Trajan's column, <https://en.wikipedia.org/w/index.php?curid=7999444>
- fig. 8: By Osado - Own work, Public Domain, <https://commons.wikimedia.org/w/index.php?curid=6505958>
- fig. 9: L'utente che ha caricato in origine il file è stato Vincent Ramos di Wikipedia in francese - Trasferito da fr.wikipedia su Commons., CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=2237975>
- fig. 10: Di Seth Schoen - Latin ATMUploaded by h-stt, CC BY-SA 2.0, <https://commons.wikimedia.org/w/index.php?curid=10908243>
- fig. 11: Pubblico dominio <https://it.wikipedia.org/w/index.php?curid=804425>

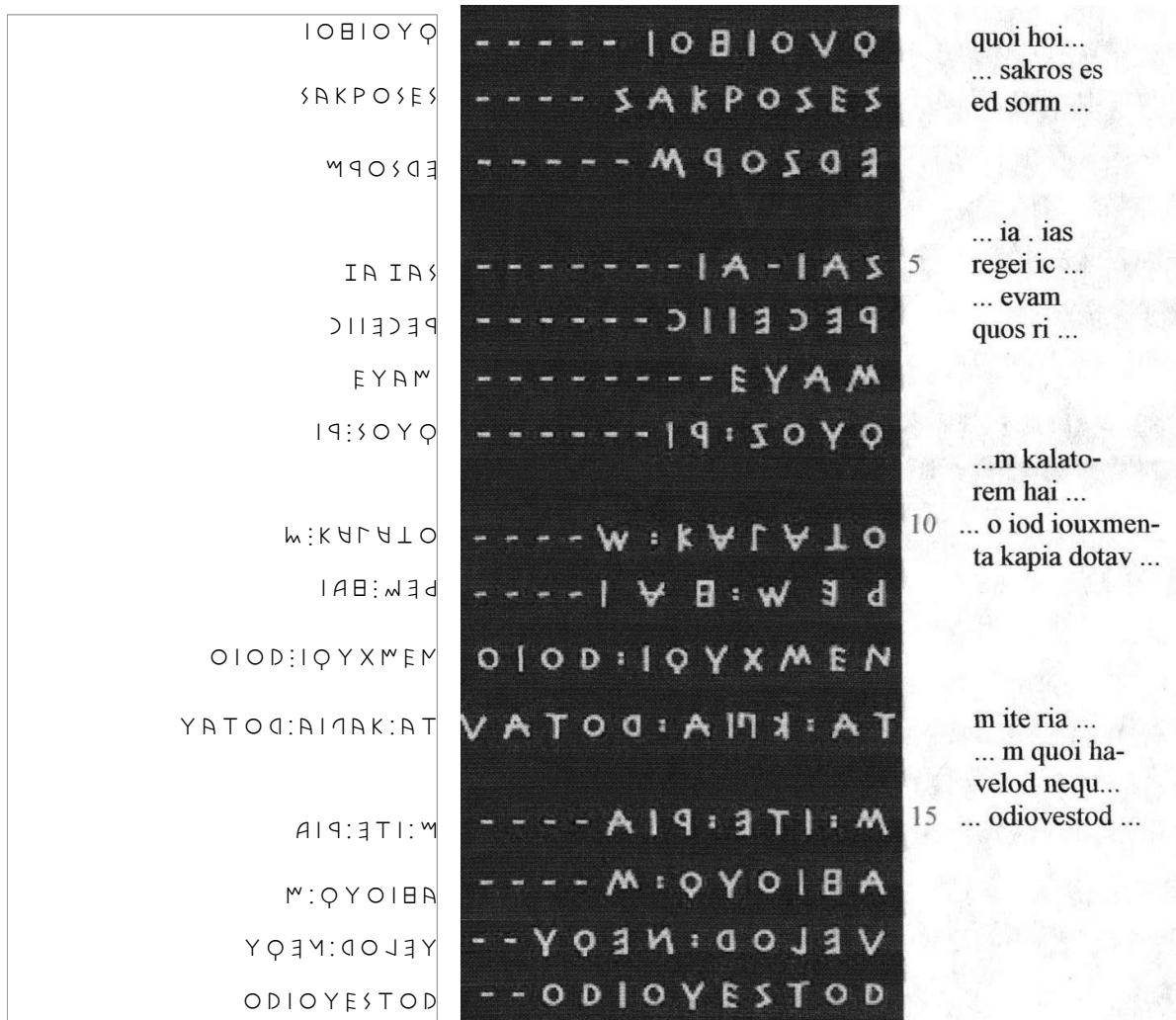


FIGURE 13: Lapis Niger typeset.

```

\startluacode
local function addmfgl(tfmdata,value)
    if value then
        local properties      = tfmdata.properties
        local parameters      = tfmdata.parameters
        local name            = tfmdata.filename
        local size             = parameters.scaledpoints
        local font, id        = fonts.constructors.readanddefine(name, size)
    for targetslot,_n in pairs(MFglyphimage) do
        local slot            = targetslot  --[==[ original glyph ]==]
        local chardata        = font.characters[slot]
    if not(chardata) then
        goto CONTINUE
    end

        properties.virtualized = true
        tfmdata.fonts          = { { id = id } }
        local width            = chardata.width or 0
        local height           = chardata.height or 0
        local depth            = chardata.depth or 0
        local correction       = number.dimenfactors.bp * width
        local mfglyph          = MFglyphimage[targetslot].img
        local mfglw            = 1
        local mfglh            = 1
        local mfglyph_depth    = MFglyphimage[targetslot].dim[3]
        if mfglyph_depth == nil then mfglyph_depth = 0 end
        if value == 'mfgl' then
            if mfglyph.width ~=0 and width ~=0 then
                mfglw = width/mfglyph.width
            else
                mfglw = 0
            end
            if (mfglyph.height+mfglyph_depth) ~=0 and (height+depth) ~=0 then
                mfglh = (height+depth)/(mfglyph.height+mfglyph_depth)
            else
                mfglh = 0
            end
        end
        if mfglw > mfglh then mfglw=mfglh else mfglh=mfglw end
        tfmdata.characters[targetslot] = {
            width      = MFglyphimage[targetslot].dim[1]*mfglw or 0 ,
            height     = MFglyphimage[targetslot].dim[2]*mfglw or 0 ,
            depth      = MFglyphimage[targetslot].dim[3]*mfglw or 0 ,
            commands   = {
                { "push" },{ 'special', 'pdf: q' },{ "special", 'pdf: 3 Tr' },
                { "slot", 1, targetslot },  --[==[ Original char ]==]
                { "right", -1*width }, { 'special', 'pdf: Q' }, { 'special', 'pdf: q' },
                { "lua", "pdf.print(' .. mfglw .. " 0 0 " .. mfglh .. " 0 0 cm \\n')" },
                { "down", 0*mfglyph_depth}, { "image", mfglyph}, { "down", -0*mfglyph_depth},
                { 'special', 'pdf: Q' },{ "pop" },
            }
        }
    }
::CONTINUE::
end
end
end

```

FIGURE 14: The code for the features `mfgl` and `mfgln`.

```

\startluacode
local _hexstr = function(s) return string.format('%x',s) end
local _hexnr = function(s) return tonumber(table.concat({'0x',s})) end
MFglyphimage = MFglyphimage or {}

local mffont = dofile('lapisniger-mfgl-2540.lua')
local characters = mffont.characters
local tfm = mffont.tfm
local dpi = mffont.dpi
for k,ch in pairs(mffont.characters) do
    if not(ch.rows) or not(ch.mfindx) then
        mffont.characters[k] = nil
    end
end
for k,ch in pairs(mffont.characters) do
    local targetslot = _hexnr(k)
    local x = ch.rows[1]:len()*4 --[==[ 8bits <-> 2 hex chars ]==]
    local y = ch.height
    local w = tfm.characters[tonumber(ch.mfindx)].width
    local h = tfm.characters[tonumber(ch.mfindx)].height
    local d = tfm.characters[tonumber(ch.mfindx)].depth
    local rows = table.concat(ch.rows,'')
    local basepoints = number.dimenfactors["bp"]
    local template = "q BI %s ID %s > EI Q"
    pdfconstant = lpdf.constant
    pdfarray = lpdf.array
    newimg = img.new
    x = tonumber(x)
    y = tonumber(y)
    w = tonumber(w)
    h = tonumber(h)
    d = tonumber(d)
    local dict = lpdf.dictionary {
        W = x,
        H = y,
        F = pdfconstant("AHx"),
        CS = nil,
        BPC = 1,
        IM = true,
        D = pdfarray({1,0}), --[==[ reverse with pdfarray({0,1}) ]==]
    }
    --[==[for some reasons it only works well if we take a lbp boundingbox (HH)]==]
    local urx, ury = 1/basepoints, 1/basepoints
    local image = newimg {
        stream = string.formatters[template](dict(),rows),
        width = w,
        height = h+d,
        bbox = { 0, 0, urx, ury },
    }
    MFglyphimage[targetslot] = {}
    MFglyphimage[targetslot].img = image
    MFglyphimage[targetslot].dim = {w,h,d}
end
\stopluacode

```

FIGURE 15: The code for the bitmap images of features `mfgl` and `mfgln`.

Primi esperimenti con node di LuaTeX

Roberto Giacomelli

Sommario

Il nuovo motore di composizione LuaTeX sta per uscire in una release stabile. Questo lavoro è un'introduzione passo passo su alcuni degli elementi interni, come i nodi `rule`, `glue` e `glyph`.

Inoltre sull'idea del contesto reale, viene sviluppato un progetto pilota per riprodurre i codici a barre nella specifica Code 39, allo scopo di svelare le potenzialità delle nuove funzioni e mettere alla prova il linguaggio di programmazione Lua.

Abstract

The new LuaTeX typesetting engine is going to be released in a stable version, ready for powerful applications. This paper is a step-by-step introduction of several LuaTeX internals such as the `rule`, `glue`, and `glyph` nodes.

Moreover, a pilot project is developed as a real design problem, to reproduce barcodes in the Code 39 specification, in such a way to reveal the new features and to dive into the Lua programming language.

1 Introduzione

Il nuovo motore di composizione LuaTeX offre una semplice modalità di scambio dati tra il codice Lua eseguito durante la compilazione del sorgente e il compositore TeX. In dettaglio, la funzione `tex.print()` consente d'inserire nel flusso di token qualsiasi elemento si voglia per farlo poi elaborare ulteriormente da TeX come se lo avessimo digitato manualmente nel sorgente.

Come esempio minimo compilabile di questa *comunicazione*, scriviamo nella macro primitiva espandibile `\directlua` il seguente codice Lua per immettere un testo in maiuscolo:

```
\directlua{
  local str = string.upper("casa")
  tex.print(str)
}
\bye
```

La prima cosa da notare è che questo sorgente non è scritto secondo il linguaggio di un formato utente come L^ATeX o ConTeXt, ma nella sintassi diretta del motore di composizione LuaTeX. Esattamente come per TeX, in esso non esiste un preambolo e nemmeno la nozione di ambiente. L'unica macro richiesta è `\bye`¹ perché necessaria all'uscita

1. Questa macro non è primitiva ma è definita in Plain tramite l'istruzione `\end`.

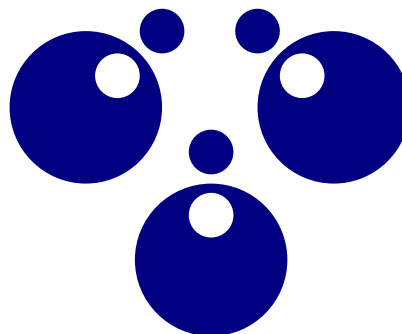


FIGURA 1: Il logo del progetto LuaTeX (logo by Taco Hoeackwater & Hans Hagen, used by permission).

del programma che altrimenti entrerebbe in attesa al raggiungimento della fine del file.

Inserendo il codice in un file di nome `test.tex` è possibile compilarlo per verificarne il risultato con il comando da terminale:

```
$ luatex test
```

Effettivamente ha funzionato: il codice argomento della macro `\directlua` è stato eseguito dall'interprete Lua interno a LuaTeX e i token della parola CASA sono stati inviati a TeX, che ne ha poi composto l'unico capoverso del documento.

Questa modalità consente facilmente di sfruttare il linguaggio di programmazione Lua per elaborare informazioni e farle poi comporre da TeX. Ciò è davvero notevole se pensiamo che con Lua è possibile connettersi a database o compiere calcoli in virgola mobile, tutti compiti praticamente impossibili con il motore tradizionale.

Ma è naturale chiedersi se esiste un modo più diretto interno a Lua con cui svolgere *direttamente* compiti di composizione. La risposta è sì, grazie alla libreria `node` di LuaTeX; illustrarne piacevolmente alcune caratteristiche di base è lo scopo di questo lavoro.

1.1 Struttura dell'articolo

L'articolo si compone di due parti: la prima è un'introduzione alla libreria `node` con esempi di codici semplici e compilabili al fine di invogliare la sperimentazione diretta da parte del lettore; la seconda è un approfondimento applicativo che mostra i possibili percorsi di un progetto ipotetico per la stampa di codici a barre, con il duplice obiettivo di mostrare le nuove potenzialità di LuaTeX e alcune tecniche di programmazione implementabili in Lua.

1.2 Il linguaggio Lua

Ci sono alcune guide sintetiche disponibili in rete, ma per imparare a programmare in Lua il riferimento più autorevole è certamente il libro scritto proprio dallo sviluppatore principale di questo linguaggio dal titolo “Programming in Lua” (IERUSALIMSKY, 2013), meglio noto semplicemente come PIL.

I concetti fondamentali da acquisire sono:

- caratteristiche del sistema dinamico dei tipi;
- regole di binding dell’assegnamento;
- basi del tipo *tabella*, l’unico dato strutturato predefinito, e del suo costruttore;
- sintassi dei costrutti di controllo;
- iteratori;
- funzioni come tipi di prima classe.

Studiare il linguaggio invece di tentare d’indovinarne la sintassi vi darà solide basi facendovi risparmiare molto tempo.

1.3 Il motore LuaTEX

Semplificando, possiamo pensare a LuaTEX come il motore di composizione TEX a cui è stato aggiunto un interprete Lua. In qualsiasi momento della compilazione è possibile far eseguire codice Lua che al termine ripasserà il controllo a TEX.

Ma Lua in TEX sarebbe del tutto inutile se non ci fosse un modo per scambiare dati tra i due livelli d’esecuzione. In effetti potremmo far elaborare i dati in modo indipendente e sequenziale in modo che l’uscita del programma Lua, o di qualsiasi altro linguaggio, sia l’ingresso dati per il compositore.

Un report, per esempio, potrebbe essere costruito con un programma che prima interroghi la base dati e poi scriva il file di testo pronto per essere compilato con TEX².

Come vedremo in seguito, lo scambio dati da Lua a TEX non solo è possibile, ma la tecnologia dei nodi costituisce una funzionalità notevole e a quanto pare unica tra tutti i programmi tipografici, anche se ancora *work in progress* e a portata di crash.

2 I nodi

In termini generali, il funzionamento del motore di composizione TEX si compone di quattro livelli in cascata:

2. L’idea è stata ampiamente sviluppata nell’articolo apparso sul numero scorso di *ArsTEXnica* (GIACOMELLI e PIGNALBERI, 2015). In questo lavoro è anche possibile reperire informazioni utili sul linguaggio Lua, in particolare sul formato *data description* cioè sull’idea di usare il costruttore di tabelle per creare un archivio dati testuale interpretabile direttamente da Lua.

Input processor: traduzione del flusso di caratteri d’ingresso nella lista di token;

Expansion processor: sostituzione ricorsiva dei token espandibili con i token componenti;

Execution processor: esecuzione dei token non ulteriormente espandibili e invio alle liste principali orizzontali verticale e matematica;

Visual processor: creazione dei capoversi, degli allineamenti, delle pagine, composizione della matematica e scrittura del file *dvi*.

I dati d’uscita di un livello diventano quelli d’ingresso del successivo fino a che i nostri sorgenti non sono trasformati in documenti di alta qualità tipografica.

In LuaTEX il codice Lua è in grado d’inserire dati in due dei processi descritti: nella forma di token nel processo d’espansione e nella forma di nodi nel processo di esecuzione.

Abbiamo già incontrato nell’introduzione la funzione `tex.print()` all’interno di una macro `\directlua`. In esecuzione, questa funzione rilascia l’argomento come informazione tradotta in token. Ciò avviene per esempio con il sorgente seguente:

```
\def\centra#1{\directlua{
  local s = string.upper("#1")
  local m = "\string\centerline{...s...}"
  tex.print(m)
}}
\centra{casa}
\bye
```

un modo complicato di scrivere il codice TEX `\centerline{CASA}` attraverso l’espansione della primitiva `\directlua` poi ulteriormente espanso al passaggio successivo dal processo di espansione.

La presenza della macro `\string` è solo un modo per produrre il testo corretto dopo la lettura del contenuto della macro `\directlua`. Ciò naturalmente non sarebbe necessario se il codice Lua fosse richiamato da un file esterno. Questa lettura permette di passare dati a Lua proprio come si vede nell’esempio dove il comando `\centra` attende un argomento che sarà sostituito all’interno del codice Lua alla riga 2 prima dell’esecuzione.

Più in dettaglio, tutti i caratteri passati alle funzioni `tex.print()` e simili vengono mano a mano inseriti in un’area di memoria e rilasciati nel flusso d’espansione come risultato dell’espansione della macro `\directlua`.

Con la libreria `node` di LuaTEX invece inseriamo non caratteri in fase di espansione, ma elementi tipografici completi direttamente nelle liste orizzontale, verticale o matematica del compositore al terzo livello di elaborazione, già pronti per essere disposti sulla pagina nel processo visuale.

2.1 Nodi rule

Riferendoci al manuale della versione beta 0.80³ di Lua $\text{\TeX}$, ovvero la versione inclusa in TeX Live 2015, alla libreria **node** viene dedicato il capitolo 4, che elenca le funzioni disponibili, e il capitolo 8, che elenca i tipi di nodi con i singoli parametri. Tra i vari tipi di nodi troviamo il tipo **rule** che ci permetterà di fare i primi semplici esperimenti.

Un nodo **rule** rappresenta un rettangolo pieno che corrisponde esattamente a quelli prodotti con la macro `\rule` di L $\text{\TeX}$. Per costruire un filetto di 10pt di larghezza e 5pt di altezza, con la macro `\rule` scriviamo:

```
\rule{10pt}{5pt}
```

e il risultato è .

Con la libreria **node** è possibile ottenere la stessa cosa programmando in Lua. Il codice è abbastanza semplice: prima si crea un nuovo oggetto **node** di tipo **rule** passando semplicemente la stringa del nome alla funzione `node.new()`, poi si impostano le dimensioni in punti scalati con assegnazione diretta ai campi dimensionali e infine si immette l'oggetto composto nel flusso di uscita con la funzione `write()`:

```
A rule 10pt width and 5pt height:
\directlua{
local r = node.new("rule")

r.width  = 10*65536
r.height = 5*65536

node.write(r)
}.
\bye
```

Compilando questo sorgente si ottiene il risultato atteso perché, senza esserne consapevoli, abbiamo scritto il nodo quando TeX si trovava nel modo orizzontale all'interno di un capoverso. Se non ci si trovasse in **hmode**, come quando siamo all'inizio di un nuovo capoverso, accadrebbe l'inatteso: per verificarlo basta inserire nel codice precedente una riga vuota tra la prima e la seconda.

Una volta inseriti i nodi, per assicurarci che si trovino in modo orizzontale possiamo inserire la macro `\leavevmode` prima del codice Lua oppure impacchettare il nodo all'interno di un nodo che rappresenta un box, come vedremo in seguito nel dettaglio.

2.2 I punti scalati

Prima di passare al prossimo oggetto **node**, soffermiamoci un momento sulle unità di misura dimensionali dei *punti scalati* **sp**. TeX opera internamente in aritmetica intera e converte in punti scalati

3. Ad oggi, la versione più recente di Lua $\text{\TeX}$, la 0.89.0 datata 4 febbraio 2016, non è disponibile ovviamente in TeXLive ma nei repository di codice del progetto, oppure installando ConTeXt minimal <http://wiki.contextgarden.net/ConTeXt-Standalone>.

TABELLA 1: Fattori interi di conversione tra le unità di misura delle lunghezze accettate dal motore di composizione TeX e l'unità fondamentale dei punti scalati, ricavati con la funzione `tex.sp()` di Lua $\text{\TeX}$.

Simbolo	Unità di misura	Fattore
cm	centimetro	1 864 679
mm	millimetro	186 467
in	pollice	4 736 286
pt	punto tipografico	65 536
pc	pica	786 432
dd	punto Didot	70 124
cc	cicero	841 489
bp	punto grande	65 781
sp	punto scalato	1

le misure espresse in una delle altre otto unità di misura ammesse, per esempio dai punti **pt**, dai pollici **in**, dai millimetri **mm** o dai centimetri **cm**.

Un punto scalato **sp** equivale esattamente a una parte su 65 536 di un punto tipografico **pt** — una distanza veramente molto piccola — ed è l'unità fondamentale di TeX, quindi anche di Lua $\text{\TeX}$. Perciò quando si lavora con la libreria **node** le dimensioni devono essere moltiplicate per i fattori riportati nella tabella 1 per la conversioni necessarie verso i punti scalati.

Nella libreria **tex** di Lua $\text{\TeX}$ esiste la funzione `sp()` che accetta una stringa nel formato `<numero><unità>`. Nel codice precedente avremo potuto quindi scrivere:

```
r.width  = tex.sp "10pt"
r.height = tex.sp " 5pt"
```

Questa funzione è interessante perché rende il codice più comprensibile e può essere impiegata per convertire un dato introdotto direttamente dall'utente, per esempio quando egli impiega una macro che richiede di specificare una lunghezza. Naturalmente l'uso diretto di fattori di conversione è più efficiente e lascia allo sviluppatore qualche libertà in più rispetto alla precisione di conversione.

2.3 Nodi glue

Occupiamoci ora di un secondo tipo di nodo che rappresenta uno spazio tra due elementi: il tipo **glue**. Paradossalmente lo spazio vuoto in TeX è molto più complicato di quello che si potrebbe a prima vista immaginare. Non è infatti solo una dimensione rigida, ma ammette una variazione dinamica in funzione delle esigenze compositive come una sorta di molla. Questa *elasticità* è definita da due misure, una per il massimo accorciamento chiamata **shrink** e una per il massimo allungamento chiamata **stretch**.

Per creare un nodo **glue** lo schema è identico al precedente:

1. creazione del nodo con la funzione di libreria `node.new(<type name>)`;

2. impostazione di parametri interni dell'oggetto;
3. scrittura in uscita dell'oggetto verso il motore di composizione.

Nella versione attuale di LuaT_EX le proprietà elastiche di uno spazio si impostano attraverso un ulteriore oggetto nodo chiamato `glue_spec`, cosa che nel futuro potrebbe cambiare con l'unione dei due tipi. Un esempio di codice è il seguente, nel quale si lascia uno spazio di 18pt — rigido — tra due gruppi di lettere:

```
ABC\directlua{
  local spec18pt = node.new("glue_spec")
  spec18pt.width = 18*65536

  local space = node.new("glue")
  space.spec = spec18pt

  node.write(space)
}DEF
\bye
```

2.4 Interazione con la lista T_EX

Prendendo spunto dal codice precedente, se voleste inserire due spazi consecutivi identici attraverso i nodi, cosa scrivereste?

Prima di proseguire nella lettura pensateci un momento e magari provate a compilare un vostro esempio. La mia prima soluzione — sbagliata — per risolvere questo problema è stata quella di duplicare la chiamata della funzione `write()` con lo stesso oggetto nodo, ritrovandomi così con il programma bloccato:

```
ABC\directlua{
  local space = node.new("glue")
  space.spec = node.new("glue_spec")
  space.spec.width = 18*65536

  % errore!
  node.write(space)
  node.write(space)
}DEF
\bye
```

Per capire cosa è avvenuto occorre ricordarsi che T_EX processa gli elementi tipografici inserendoli in una lista, una struttura dati dove ogni elemento mantiene un riferimento al successivo.

Come recita il manuale di LuaT_EX, la funzione `write()` — peraltro ancora sperimentale — copia in coda alla lista di T_EX solamente il riferimento al nodo e non l'intero oggetto. Quando T_EX elabora il primo nodo non riesce quindi a scorrere la lista perché il nodo successivo è ancora un riferimento allo stesso nodo.

Questo ci insegna che con la libreria `node` è molto facile corrompere la lista principale usata da T_EX per comporre gli elementi tipografici. Con le versioni stabili di LuaT_EX i messaggi d'errore e la documentazione potranno aiutare gli sviluppatori di applicazioni a divenire esperti evitando questo genere di errori.

2.5 Collegare insieme i nodi

A questo punto non vi sorprenderà sapere che anche i nodi di LuaT_EX sono in realtà potenziali elementi di una lista. Più precisamente, invece di elaborare i nodi in modo indipendente uno dall'altro, li si collega in una lista e la si invia a T_EX indicando solamente il primo nodo.

Per provare come possa apparire il codice diamoci un nuovo problema: disegnare dieci quadrati di 10pt di lato a distanza di 10pt uno dall'altro. Sapendo che ogni oggetto nodo ha un campo `next` in cui memorizzare il riferimento al nodo successivo, provate a prendere carta e matita per scrivere la vostra soluzione a questo esercizio.

Chiudete però le pagine della rivista perché tra un attimo riporterò il codice scritto da me — incurante dei pericoli — con cui potrete confrontare le vostre soluzioni per ottenere questo simpatico disegno:



```
\leavevmode\directlua{
  local tenpt = 10 * 65536
  local firstElem = node.new("rule")
  firstElem.width = tenpt
  firstElem.height = tenpt

  local lastElem = firstElem

  for _ = 2,10 do
    local s = node.new("glue")
    s.spec = node.new("glue_spec")
    s.spec.width = tenpt
    lastElem.next = s
    lastElem = s

    local q = node.new("rule")
    q.width = tenpt
    q.height = tenpt
    lastElem.next = q
    lastElem = q
  end

  node.write(firstElem)
}
\bye
```

Quello che ho fatto è impostare manualmente i campi `next` dei nodi per *appendere* uno alla volta i successivi alla lista, mantenendo un riferimento all'ultimo e al primo nodo da passare alla funzione `write()`. Tuttavia, è certamente più sicuro formare la lista con le funzionalità interne di LuaT_EX sia per maggiore compatibilità con le versioni future sia perché i nodi, avendo anche il campo `prev`, dovrebbero formare liste doppiamente concatenate.

Ci viene in aiuto la funzione `insert_after()` della libreria `node`, in grado di aggiungere un nuovo elemento a una lista in base alla posizione di uno dei suoi nodi. Gli argomenti previsti da questa funzione sono il riferimento al primo nodo della

lista, il riferimento al nodo dopo il quale verrà inserito il nuovo elemento e il riferimento al nuovo nodo. I parametri restituiti sono due: il riferimento al primo nodo della lista e quello al nuovo nodo aggiunto.

Questa funzione — assieme a `insert_before()` — può anche svolgere il ruolo di *costruttore* della lista, perché quando il riferimento al nodo di testa assume il valore `nil`, valore assunto anche dal secondo parametro, varrà il caso di *lista vuota*.

Volendo usare queste funzioni, si dovrà però fare attenzione a che il riferimento al nodo subito dopo il quale si vuole inserire quello nuovo appartenga alla stessa lista individuata dal nodo primo argomento. Il codice può essere riscritto così:

```
\leavevmode\directlua{
local tenpt = 10 * 65536
local elem = node.new("rule")
elem.width = tenpt
elem.height = tenpt

% nuova lista
local head, last = node.insert_after(
    nil, nil, elem
)

for _ = 2,10 do
    local s = node.new("glue")
    s.spec = node.new("glue_spec")
    s.spec.width = tenpt
    head, last = node.insert_after(
        head, last, s
    )

    local q = node.new("rule")
    q.width = tenpt
    q.height = tenpt
    head, last = node.insert_after(
        head, last, q
    )
end

node.write(head)
}
\bye
```

La versione finale che darò del codice d'esempio è ancora più compatta. Essa fa uso della funzione `node.copy()`, che duplica il nodo argomento compresi gli eventuali sotto-nodi e fa a meno del riferimento all'ultimo nodo della lista⁴ che in precedenza avevo chiamato `last`, inserendo ogni volta i nodi quadrato e spazio con ordine inverso subito dopo il nodo di testa:

```
\leavevmode\directlua{
local tenpt = 10 * 65536

% square node
```

4. Se non siete a vostro agio con la semantica che Lua riserva alla corrispondenza degli argomenti di input e output delle funzioni consultate una guida o il libro IERUSALIMSKY (2013).

```
local q = node.new("rule")
q.width = tenpt
q.height = tenpt

% space node
local s = node.new("glue")
s.spec = node.new("glue_spec")
s.spec.width = tenpt

% a new list
local head = node.insert_after(nil, nil, q)

for _ = 2,10 do
    head = node.insert_after(
        head, head, node.copy(q)
    )
    head = node.insert_after(
        head, head, node.copy(s)
    )
end

node.free(s)
node.write(head)
}
\bye
```

Se il numero dei quadrati fosse una potenza di 2 si potrebbe a ogni ciclo appendere alla lista una copia di se stessa tramite la funzione `node.copy_list()`. Lascio al lettore altri interessanti esperimenti dandogli appuntamento per la discussione sul forum del Q_TT.

2.6 Nodi in memoria

Il sistema dei tipi di Lua è estensibile, ciò significa che esiste un meccanismo per aggiungere nuovi tipi chiamati *userdata* implementandoli in linguaggio C. Ebbene, i nodi sono oggetti *userdata* compatibili a quelli prodotti da T_EX per le sue liste.

In Lua l'oggetto nodo è un riferimento, diremmo un puntatore, all'oggetto *userdata* in memoria. Una volta inserito all'interno delle liste svuotate dal Visual processor, il codice Lua non può più eliminarlo dalla memoria automaticamente tramite il Garbage Collector⁵ perché potrebbe eliminare un oggetto non ancora processato. In altre parole, Lua non ha e non può avere il controllo sull'operato di T_EX, almeno con la tecnologia sviluppata con l'attuale LuaT_EX.

Per questo motivo i nodi sono stati esclusi dal processo automatico del Garbage Collector di Lua, perciò la gestione della memoria deve essere fatta manualmente dallo sviluppatore.

In definitiva, le regole da seguire si basano sul fatto che non è possibile per Lua conoscere lo stato

5. Il Garbage Collector è un processo automatico attivo in fase di esecuzione che libera la memoria non più indirizzata da almeno un riferimento. Questo succede per esempio quando assegniamo il valore `nil` all'unica variabile Lua che indirizza una tabella, perché da quel momento essa non può più essere utilizzata nel programma e può quindi essere eliminata.

di un oggetto in memoria che è stato ormai incluso nelle liste di T_EX con `node.write()`, il ché significa che non possiamo sapere se l'oggetto è ancora utilizzato da T_EX perché si trova nella lista in attesa di essere inviato al processo di visualizzazione, né sapere se T_EX stesso ha già provveduto a liberare la memoria dell'oggetto perché ormai già processato.

Forse non vi sarà passata inosservata una delle ultime righe di codice dell'esempio precedente, quella dove viene chiamata la funzione `node.free()`. Farlo è corretto in questo caso perché quel nodo `s` è sempre e solo stato copiato, non fa parte di nessuna lista e non è mai stato inviato al motore di composizione.

2.7 Nodi glyph

Uno dei tipi di nodi che ci si aspetterebbe di trovare è quello che rappresenta i caratteri. Finalmente possiamo ora dedicarci al tipo chiamato `glyph`. Tra i campi dell'oggetto, quelli fondamentali sono due: `char` con la codifica del carattere e `font`.

Sapendo questo, proviamo a scrivere il codice Lua per stampare tutte le lettere maiuscole dell'alfabeto latino. Al solito il consiglio è di non proseguire nella lettura prima di aver provato in proprio con i suggerimenti di usare la codifica ASCII e la funzione `font.current()` per ottenere il font da usare. Ecco una possibile soluzione:

```
\leavevmode\directlua{
local alpha, last
for i=1,26 do
  local c = node.new("glyph")
  c.char = 64+i
  c.font = font.current()
  alpha, last = node.insert_after(
    alpha, last, c
  )
end

node.write(alpha)
}\bye
```

Sì, funziona, perché la codifica ASCII pur non essendo quella di LuaT_EX che opera in Unicode, è inclusa nella trasformazione multi-byte UTF-8 di quest'ultima fino alla posizione numero 127.

A questo punto dovremo parlare di font, legature e tanti altri temi legati alla composizione del testo, ma a mio parere sono argomenti da affrontare solo dopo aver compreso il funzionamento di base dei nodi, che del resto si legano a tecniche avanzate come gli *attributi* e le funzioni di *callback*. Proseguiamo dunque con il prossimo tipo di nodo che è infatti fondamentale per le applicazioni tipografiche: la scatola.

2.8 Nodi scatole

Due funzioni della libreria `node` consentono di costruire scatole con al loro interno elementi impacchettati in orizzontale, funzione `hpack()`, o in

verticale, funzione `vpack()`. Esse accettano una lista di nodi tramite il riferimento al nodo di testa e restituiscono il riferimento al nodo del box corrispondente.

È dunque indispensabile conoscere cosa sono i box in T_EX, quali i dettagli dei modi di composizione, delle dimensioni e degli allineamenti; del resto questo è stato vero fin dall'inizio. Possiamo riassumere il tutto con la frase *non c'è Lua senza T_EX...* almeno quando si parla di LuaT_EX.

Altra conseguenza implicita dell'idea di contenitore è l'annidamento. Possiamo cioè inserire un box dentro l'altro impacchettando oggetti fino a raggiungere il risultato tipografico desiderato. Ne mostrerò esempi e applicazioni già nella prossima sezione.

Studiamo questo esempio:

```
\directlua{
local function glyph(c)
  local n = node.new("glyph")
  n.char = c
  n.font = font.current()
  return n
end

local function boxed(s)
  local h, l
  for c in string.utfvalues(s) do
    h, l = node.insert_after(
      h, l, glyph(c)
    )
  end
  return node.hpack(h)
end

local b1 = boxed "ABC"
local b2 = boxed "123456789"
local b3 = boxed "DEF"
b1.next = b2
b2.next = b3

local b = node.vpack(b1)
tex.box[0] = b
}
% stampo e svuoto il registro 0
\box0
\bye
```

un sorgente compilabile dove è prodotto un box verticale da una lista di nodi con tre scatole orizzontali contenenti a sua volta altrettante liste di glifi, con il seguente risultato:

```
ABC
123456789
DEF
```

Nel codice troviamo anche la modalità con cui si possono usare i registri box direttamente da T_EX: è sufficiente assegnare a un registro il nodo iniziale della lista tramite indicizzazione diretta dell'array `tex.box`. Possiamo dunque usare i comandi T_EX previsti per la composizione, come `\box` che stampa e libera il contenuto del registro dal numero che

lo segue — con a disposizione ben 65 536 registri — e `\copy` che stampa il contenuto ma lascia intatto il registro.

Il fatto che la tabella/array box della libreria `tex`, quindi un oggetto Lua, sia così intimamente legato ai registri `TEX`, come abbiamo appena visto, è un chiaro indice del funzionamento interno dei nodi.

È anche possibile utilizzare i registri allocati da `TEX` tramite una control sequence per evitare che un pacchetto collida con un altro nell'indirizzarli per numero. Si tratta di rispettare una convenzione dei nomi già in uso nei pacchetti `LATEX` — una sorta di *namespace* basato sul simbolo `@` — per rendere molto meno probabile la sovrapposizione di macro e registri.

Occorre prima allocare il registro con la primitiva `\newbox` e poi assegnarlo con la funzione `tex.setbox()` in Lua. Vediamone un esempio rimandando al manuale di `LuaTEX` per i dettagli della sintassi:

```
\newbox\mypackMybox

\directlua{
local function glyph(c)
    local n = node.new("glyph")
    n.char = c
    n.font = font.current()
    return n
end

local function boxed(s)
    local h, l
    for c in string.utfvalues(s) do
        h, l = node.insert_after(
            h, l, glyph(c)
        )
    end
    return node.hpack(h)
end

local b = boxed "ABC"
tex.setbox("mypackMybox", b)
}

\copy\mypackMybox
\copy\mypackMybox
\box\mypackMybox
\bye
```

Accanto alla funzione `setbox()` esiste anche `getbox()` che accetta l'identificatore del registro box, sia esso il numero intero o la stringa della control sequence, per restituire il nodo interno al contenitore. Ciò lascia supporre, e ne lascio la verifica al lettore, che sia possibile modificare il contenuto di un registro creato in precedenza con le macro di `TEX`.

Prima di passare oltre consideriamo questo frammento di codice `TEX`:

```
\hbox to 64pt {A\hfil A}
```

Esso compone una scatola di larghezza fissata a 64pt contenente una grandezza elastica il cui effetto è per così dire di *spingere* le due lettere verso l'esterno fino alle pareti del contenitore. Più è elevato il valore di stretching dello spazio elastico e maggiore sarà l'effetto. Possiamo fare lo stesso con i nodi in Lua?

La risposta è sì. Occorre fornire alla funzione `hpack()` non solo il nodo che rappresenta la lista, ma altri due argomenti che finora vi ho tenuto nascosti. Il primo è una dimensione e il secondo è una stringa che può essere `"exactly"` — nel qual caso la dimensione sarà interpretata come la lunghezza del box — oppure `"additional"`, con la lunghezza che sarà la capacità di allungarsi del box stesso.

Proviamo a ricreare gli stessi nodi prodotti da `TEX` con la breve riga del codice precedente:

```
\directlua{
% glyphs
local a = node.new "glyph"
a.char, a.font = 65, font.current()
local a1 = node.copy(a)

% skip
local skip = node.new "glue"
skip.spec = node.new "glue_spec"
skip.spec.stretch = tex.sp "60pt"

% list it together
a.next, skip.next = skip, a1

% packed it in a hbox
local box, b = node.hpack(
    a, tex.sp "64pt", "exactly"
)
tex.print("box badness:", b)
tex.box[0] = box
}

$|$\copy0$|$
$|$\copy0$|$

$|$\hbox to 64pt{A\hfil A}$|$
$|$\hbox to 64pt{A\hfil A}$|$
\bye
```

La verifica ha successo anche nel mostrare che la funzione `hpack()` fornisce un secondo parametro che misura la qualità della spaziatura del box chiamata *badness*, un intero compreso tra 0 e 10 000 — che nel nostro caso vale 54.

3 Applicazione barcode

Ci proponiamo di utilizzare la tecnologia dei nodi di `LuaTEX` per stampare il barcode di una stringa testuale secondo il formato Code 39. Si tratta di un'esercitazione interessante perché implica un certo sforzo di progettazione e che illustrerò passo passo nelle prossime sezioni cominciando al solito da una definizione generale.

Il *barcode* è un disegno di rettangoli pieni che non si sovrappongono costruito seguendo regole di codifica e decodifica di un testo e diffuso da enti o industrie.

Una scacchiera risponde ai requisiti grafici ma non è un barcode, non essendo conforme a nessuno dei formati noti e riconosciuti dai lettori ottici o a luce laser. Se lo fosse, al termine della scansione otterrei la sequenza dei caratteri rappresentati. Lo è invece la figura che segue, trattandosi del QR Code che scansionato con i vostri smartphone vi porta al sito del GfIr:



Il più semplice formato monodimensionale Code 39 Wikipedia risale al 1974 a opera del Dottor David Allais e di Ray Stevens allora alla Intermec, un produttore di scanner americano. Esso consiste in una sequenza di barre della stessa altezza e di spessore sottile o spesso che chiameremo rispettivamente *b* e *B*, distanziate l'una dall'altra da spazi sottili o spessi che chiameremo rispettivamente *w* e *W*. Un esempio è il seguente che codifica lo stesso testo del precedente barcode:



Questi strani disegni geometrici, ormai talmente diffusi da non attirare più la nostra attenzione, nascondono ragionamenti matematici piuttosto complessi allo scopo di rendere la scansione del codice più affidabile possibile. Con considerazioni come queste il Code 39 assegna a un insieme di 43 caratteri — 10 cifre numeriche, 26 lettere maiuscole e 7 segni d'interpunzione — un'univoca rappresentazione grafica con 5 barre distanziate da 4 spazi a due diversi spessori, con 3 elementi spessi e 6 sottili.

Un simbolo particolare a codifica asimmetrica, l'asterisco, apre e chiude la sequenza delle barre del testo codificato; lo si può verificare confrontando i caratteri stampati sotto al barcode precedente.

3.1 Il carattere A

L'idea è quella di stampare la rappresentazione della lettera A — codificata con la sequenza *BwbwbWbwB* — con la funzione Lua che chiameremo *bar_char()* che accetti la stringa di codifica del carattere e ne restituisca il primo nodo della lista dei 9 elementi *rule* e *glue*.

Ciò è ragionevole, ma come reperire le misure degli elementi? Le opzioni sono tre: inserire le misure nel corpo della funzione, negli argomenti o in una variabile globale. Per il momento scegliamo la soluzione più semplice, ovvero la prima:

```
\leavevmode\directlua{
local function bar_char(code)
    % 1D code39 barcode
    % bar definitions:
    % b = narrow black bar
    % B = wide black bar
    % w = narrow white bar
    % W = wide white bar
    % module = 0.25mm
    % height = 8mm
    local mod = 0.25 * 186468
    local b, w = mod, mod
    local B, W = 3*mod, 3*mod
    local height = 8 * 186468

    local head, last
    for c in string.gmatch(code, ".") do
        if c == "b" then
            local bar_b = node.new("rule")
            bar_b.width = b
            bar_b.height = height
            head, last = node.insert_after(
                head, last, bar_b
            )
        elseif c == "B" then
            local bar_B = node.new("rule")
            bar_B.width = B
            bar_B.height = height
            head, last = node.insert_after(
                head, last, bar_B
            )
        elseif c == "w" then
            local sp_w = node.new("glue")
            sp_w.spec=node.new("glue_spec")
            sp_w.spec.width = w
            head, last = node.insert_after(
                head, last, sp_w
            )
        elseif c == "W" then
            local sp_W = node.new("glue")
            sp_W.spec=node.new("glue_spec")
            sp_W.spec.width = W
            head, last = node.insert_after(
                head, last, sp_W
            )
        end
    end

    return head
end

% the Code 39 'A' letter
node.write(bar_char("BwbwbWbwB"))
}
\bye
```

L'esecuzione di questo codice fornisce il disegno di figura 2 con cui possiamo verificare la correttezza di spessore e spaziatura. Il nucleo della funzione generatrice consiste in un'istruzione condizionale a più rami poiché Lua non dispone di un costrutto *switch*, ma in compenso lo si può migliorare in molti modi, per esempio sfruttando il fatto che le funzioni Lua sono tipi di prima classe.

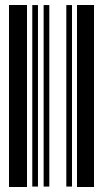


FIGURA 2: Rappresentazione della lettera A nel formato Code 39 ottenuta con il codice riportato nel testo a partire dalla stringa 'BwbwbWbwB'. Non si tratta ancora di un barcode completo perché mancano i caratteri asterisco di inizio/fine.

Ciò rende possibile associare alle quattro chiavi di formato `b`, `B`, `w` e `W`, le funzioni che generano il nodo corrispondente:

```
local mod = 0.25 * 186468
local b, w = mod, mod
local B, W = 3*mod, 3*mod
local height = 8 * 186468

local Efactory = {
  b = function ()
    local bar = node.new("rule")
    bar.width = b
    bar.height = height
    return bar
  end,
  w = function ()
    local white = node.new("glue")
    white.spec = node.new("glue_spec")
    white.spec.width = w
    return white
  end,
  B = function ()
    local bar = node.new("rule")
    bar.width = B
    bar.height = height
    return bar
  end,
  W = function ()
    local white = node.new("glue")
    white.spec = node.new("glue_spec")
    white.spec.width = W
    return white
  end,
}
```

e riscrivere `bar_char()` così, indicizzando la tabella e lanciando la chiamata della funzione restituita con una coppia di parentesi tonde:

```
local function bar_char(code)
  local head, last
  for c in string.gmatch(code, ".") do
    head, last = node.insert_after(
      head, last, Efactory[c]()
    )
  end

  return head
end
```

Questo codice mostra anche un'altra caratteristica di Lua: le *closure*. Invece di entrare nei

dettagli dirò solamente dove nel codice entrano in azione, cioè all'interno delle funzioni `factory` per la costruzione dei nodi, dove si può notare che le dimensioni sono contenute in variabili che si riferiscono al contesto locale.

3.2 Di nuovo il carattere A

Nei barcode, il testo codificato può essere riportato al di sotto del disegno a barre così da consentire di recuperare manualmente l'informazione in caso di errori di lettura. Ma come aggiungere alle nostre funzioni l'opzione di riportare il simbolo immediatamente al di sotto delle barre mantenendo gli allineamenti?

Una soluzione è quella d'inserire due box all'interno di un box verticale: la scatola che contiene la sequenza di barre/spazi che codifica la lettera A e la scatola che contiene il glifo del simbolo centrato grazie a due spazi elastici di larghezza uguale a quella naturale del box superiore.

Il codice completo che implementa l'idea è il seguente:

```
\directlua{
  local module = 0.25 * 186468
  local height = 8 * 186468
  local bardim = {
    b = module,
    B = 3*module,
    w = module,
    W = 3*module,
  }

  local Efactory = {
    b = function ()
      local bar = node.new("rule")
      bar.width = bardim.b
      bar.height = height
      return bar
    end,
    w = function ()
      local white = node.new("glue")
      white.spec = node.new("glue_spec")
      white.spec.width = bardim.w
      return white
    end,
    B = function ()
      local bar = node.new("rule")
      bar.width = bardim.B
      bar.height = height
      return bar
    end,
    W = function ()
      local white = node.new("glue")
      white.spec = node.new("glue_spec")
      white.spec.width = bardim.W
      return white
    end,
  }

  local function bar_char(code)
    local len = 0
    local head, last
    for c in string.gmatch(code, ".") do
```

```

        len = len + bardim[c]
        head, last = node.insert_after(
            head, last, Efactory[c]()
        )
    end
    return head, len
end

% natural width hbox
local ABars, lenABars = bar_char(
    "BwbwbWbwb"
)
local boxABars = node.hpack(ABars)

% setting width hbox
local A_glyph = node.new("glyph")
A_glyph.char = 65
A_glyph.font = font.current()

local head_skip = node.new("glue")
head_skip.spec = node.new("glue_spec")
head_skip.spec.stretch = 6*module

local tail_skip = node.copy(head_skip)
head_skip = node.insert_after(
    head_skip, head_skip, A_glyph
)
head_skip = node.insert_after(
    head_skip, A_glyph, tail_skip
)
local boxA_glyph = node.hpack(
    head_skip, lenABars, "exactly"
)
boxA_glyph.height = 10 * 65536
boxABars = node.insert_after(
    boxABars, boxA_glyph
)

% vertical hbox assembly
local mainbox = node.vpack(boxABars)

node.write(mainbox)
}
\bye

```

La funzione `bar_char()` adesso non restituisce più solamente il nodo della lista barre/spazi, ma anche la loro larghezza naturale che servirà per dimensionare il box sottostante sommando le larghezze dei singoli elementi.

Poiché in ogni codifica di simbolo ci sono sempre 6 spessori sottili e 3 spessi, questa dimensione è pari a $6b + 3B$ solo nell'ipotesi che gli spessori delle barre coincidano con quelli degli spazi, perciò la funzione è sì più lenta ma è valida più in generale.

L'ultima osservazione che faccio tra le molte altre ancora sviluppabili è che il codice di creazione e assemblaggio dei nodi è piuttosto verboso. Un livello d'astrazione maggiore lo renderebbe compatto e molto più leggibile, facendo però perdere allo sviluppatore la costante consapevolezza di stare manipolando oggetti tipografici a basso livello, per definizione un poco *delicati*.

3.3 Gestione degli errori e dati di formato

La flessibilità d'uso della tabella di Lua ci permette di risolvere in un colpo solo sia la gestione degli errori che la rappresentazione di formato del barcode. Nella validazione del dato in ingresso ci occorrerà infatti l'informazione di esistenza del simbolo Code 39, mentre nel disegnarlo ci servirà conoscerne la sequenza corretta dei nove elementi alternati di barre e spazi.

Poiché indicizzare una tabella Lua⁶ restituirà `nil` se la chiave non esiste o al contrario il valore a essa associato, una tabella con le 43 coppie simbolo/rappresentazione contiene l'informazione necessaria e sufficiente sia a validare che a disegnare il barcode.

Un esempio di codice per quest'idea è il seguente:

```

local Code39Symbols = {
    ["0"] = "bwbWBwBwb",
    ["1"] = "BwbWbwbwB",
    ["2"] = "bWBwbwbwB",
    -- ...
}

-- validazione del carattere "0"
local char = "0"
local ok = false
if Code39Symbols[char] then
    print("carattere code39 valido")
    ok = true
end

-- stampa codice carattere se ok
if ok then
    local code = Code39Symbols[char]
    bar_char(code)
end

```

3.4 Validazione algebrica

Anche se non rigorosamente, con la tabella di Lua possiamo rappresentare un tipo di dato chiamato *tipo algebrico*, una sorta di unione del tipo enumerazione con tipi dati. Esso è molto importante nei linguaggi a tipizzazione statica che non contemplano il tipo nullo come Haskell (*Haskell*) e Rust (*Rust*).

In particolare, vorremmo usare una funzione che controlli la validità di tutti i caratteri della stringa in ingresso sulla base della codifica Code 39 e, se la stringa è valida, restituisca il dato stesso altrimenti un opportuno messaggio d'errore:

```

function validateCode39(s)
    local chars = {}
    for c in string.gmatch(s, ".") do

```

6. La tabella di Lua implementa la struttura dati chiamata *dizionario* o anche *array associativo*, l'insieme di coppie chiavi/valori. In essa possono essere memorizzati dati all'indirizzo di chiavi univoche. La sintassi del costruttore di tabelle e l'iteratore `ipairs()` del linguaggio consentono di usare l'unica struttura dati complessa di Lua come se fosse un vettore.

```

if not Code39Symbols[c] then
    local fmt = "Error: " ..
        "the char '%s' is not a "..
        "valid Code 39 symbol"
    local msg = string.format(fmt,c)
    % primo caso
    return {Err = msg}
end
chars[#chars+1] = c
end
% secondo caso
return {Some=chars}
end

```

Chiamando questa funzione otterremo una tabella contenente o la chiave **Err**, a cui sarà associato il messaggio d'errore, oppure la chiave **Some**, a cui sarà associato il vettore dei caratteri del codice. Per conoscere in quale caso ci troviamo potremo scrivere:

```

local code = validateCode39("ABC")

if code.Err then
    error(code.Err)
else
    typesetCode39(code.Some)
end

```

Questa soluzione ha il vantaggio di essere semplice ed espressiva. Ovvio che in Lua non avremo il controllo sui tipi a tempo di compilazione e che potremo chiamare direttamente `typesetCode39()` con la tabella contenente simboli non validi, per esempio con `{Some = {"a", "b", "c"}}`, ma la trovo un'idea elegante e concorde nella filosofia di un linguaggio dinamico come Lua.

3.5 Il codice completo

Dalla discussione fatta finora basata su semplici esempi di codice funzionante, il nostro progetto assumerà la seguente struttura:

- una tabella Lua conterrà le informazioni di codifica Code 39 e le funzioni utente verranno caricate come libreria esterna;
- una funzione pubblica validerà il testo in ingresso da tradurre in barcode;
- una funzione pubblica comporrà il disegno del barcode accettando solamente stringhe di testo già validate e un secondo argomento di opzioni.

Le opzioni previste per il disegno del barcode saranno solamente due: l'altezza del codice a barre — che dovrebbe essere proporzionale alla sua lunghezza — e la stampa dei caratteri codificati.

Il prossimo listato Lua è la libreria del progetto ottenuta assemblando i risultati descritti nelle sezioni precedenti. In un file esterno vengono definite funzioni e dati locali usati dall'interfaccia pubblica attraverso le closure:

```

-- libreria d'esempio per la composizione
-- di barcode base Code39

```

```

local lib = {}

```

```

local Dim = {
    mod = 0.25 * 186468, -- sp
    height = 8 * 186468, -- sp
}
Dim.b, Dim.w = Dim.mod, Dim.mod
Dim.s = Dim.mod
Dim.B, Dim.W = 3*Dim.mod, 3*Dim.mod

```

```

local Code39Symbols = {
["0"]={"b","w","b","W","B","w","B","w","b"},
["1"]={"B","w","b","W","b","w","b","w","B"},
["2"]={"b","w","B","W","b","w","b","w","B"},
["3"]={"B","w","B","W","b","w","b","w","B"},
["4"]={"b","w","b","W","B","w","b","w","B"},
["5"]={"B","w","b","W","B","w","b","w","B"},
["6"]={"b","w","B","W","B","w","b","w","B"},
["7"]={"b","w","b","W","b","w","B","w","B"},
["8"]={"B","w","b","W","b","w","B","w","B"},
["9"]={"b","w","B","W","b","w","B","w","B"},
["A"]={"B","w","b","w","b","W","b","w","B"},
["B"]={"b","w","B","w","b","W","b","w","B"},
["C"]={"B","w","B","w","b","W","b","w","B"},
["D"]={"b","w","b","w","B","W","b","w","B"},
["E"]={"B","w","b","w","B","W","b","w","B"},
["F"]={"b","w","B","w","B","W","b","w","B"},
["G"]={"b","w","b","w","b","W","B","w","B"},
["H"]={"B","w","b","w","b","W","B","w","B"},
["I"]={"b","w","B","w","b","W","B","w","B"},
["J"]={"b","w","b","w","B","W","B","w","B"},
["K"]={"B","w","b","w","b","W","b","w","B"},
["L"]={"b","w","B","w","b","W","b","w","B"},
["M"]={"B","w","B","w","b","w","W","b","B"},
["N"]={"b","w","b","w","B","w","b","W","B"},
["O"]={"B","w","b","w","B","w","b","W","B"},
["P"]={"b","w","B","w","B","w","b","W","B"},
["Q"]={"b","w","b","w","b","W","B","w","B"},
["R"]={"B","w","b","w","b","w","B","W","b"},
["S"]={"b","w","B","w","b","w","B","W","b"},
["T"]={"b","w","b","w","B","w","B","W","b"},
["U"]={"B","w","b","w","b","w","b","W","B"},
["V"]={"b","w","B","w","b","w","b","W","B"},
["W"]={"B","w","B","w","b","w","b","W","B"},
["X"]={"b","w","b","w","B","w","b","W","B"},
["Y"]={"B","w","b","w","B","w","b","w","B"},
["Z"]={"b","w","B","w","B","w","b","w","B"},
["-"]={"b","w","b","w","b","w","B","w","B"},
["."]={"B","w","b","w","b","w","B","w","B"},
[" "]={"b","w","B","w","b","w","B","w","B"},
["$"]={"b","w","b","w","b","w","w","b","B"},
["/"]={"b","w","b","w","b","w","w","b","B"},
["+"]={"b","w","b","w","b","w","w","b","B"},
["%"]={"b","w","b","w","b","w","w","b","B"},
}

```

```

local Code39Star = {
    "b","W","b","w","B","w","B","w","b"
}

```

```

local Efactory = {
    b = function ()
        local bar = node.new("rule")
    end
}

```

```

        bar.width = Dim.b
        bar.height = Dim.height
        return bar
    end,
    w = function ()
        local white = node.new("glue")
        white.spec = node.new("glue_spec")
        white.spec.width = Dim.w
        return white
    end,
    B = function ()
        local bar = node.new("rule")
        bar.width = Dim.B
        bar.height = Dim.height
        return bar
    end,
    W = function ()
        local white = node.new("glue")
        white.spec = node.new("glue_spec")
        white.spec.width = Dim.W
        return white
    end,
    s = function ()
        local white = node.new("glue")
        white.spec = node.new("glue_spec")
        white.spec.width = Dim.s
        return white
    end,
}

local function bar_char(tchars, head, last)
    for _, e in ipairs(tchars) do
        head, last = node.insert_after(
            head, last, Efactory[e]()
        )
    end
    return head, last
end

local function hboxbars(tchars)
    local head, last = bar_char(Code39Star)
    for _, c in ipairs(tchars) do
        head, last = bar_char(
            {'s'}, head, last
        )
        head, last = bar_char(
            Code39Symbols[c], head, last
        )
    end
    head, last = bar_char({'s'}, head, last)
    head, last = bar_char(
        Code39Star, head, last
    )
    return node.hpack(head)
end

local function barsdim(tcode)
    local dim = 0
    for _, e in ipairs(tcode) do
        dim = dim + Dim[e]
    end
    return dim
end

local function hboxsymb(tchars)

-- empty space under '*'
local head = node.new("glue")
head.spec = node.new("glue_spec")
head.spec.width = barsdim(Code39Star)

local last = head
for _, c in ipairs(tchars) do
    local w = barsdim(Code39Symbols[c])
    local s0 = Efactory.s()
    local g1 = node.new("glue")
    g1.spec = node.new("glue_spec")
    g1.spec.stretch = w/2
    local g2 = node.copy(g1)
    local glyph = node.new("glyph")
    glyph.char = string.byte(c)
    glyph.font = font.current()
    g1 = node.insert_after(g1, g1, g2)
    g1 = node.insert_after(g1, g1, glyph)
    local hb=node.hpack(g1, w, "exactly")
    head, last = node.insert_after(
        head, last, s0
    )
    head, last = node.insert_after(
        head, last, hb
    )
end
local box = node.hpack(head)
box.height = tex.sp "10pt"
return box
end

-- public function
function lib.validate(s)
    local chars = {}
    for c in string.gmatch(s, ".") do
        if not Code39Symbols[c] then
            local fmt = "Error: " ..
                "the char '%s' is not a valid"..
                " Code 39 symbol"
            local msg = string.format(fmt,c)
            return {Err = msg}
        end
        chars[#chars+1] = c
    end
    return {Some=chars}
end

function lib.typeset(tcode, topt)
    if tcode.Err then
        error(tcode.Err)
    end
    local symb = false
    local olddim
    if topt then
        -- optional barcode height
        if topt.h then
            olddim = Dim.height
            Dim.height = tex.sp(topt.h)
        end
        -- optional symbol ref
        if topt.symb then
            symb = true
        end
    end
end

```

```

end

local barbox = hboxbars(tcode.Some)
if symb then
    local symbbox = hboxsymb(tcode.Some)
    barbox = node.insert_after(
        barbox, barbox, symbbox
    )
end
local vbox = node.vpack(barbox)
local res = node.hpack(vbox)
node.write(res)

if olddim then
    Dim.height = olddim
end
end

return lib

```

3.6 Test

È sempre fondamentale verificare la correttezza del programma con una batteria di test il più completa e automatica possibile. Nel caso della libreria Lua_T_E_X, per stampare barcode secondo il formato Code 39, i test sono stati di due tipi: controllo a schermo del disegno del barcode e verifica fisica mediante lettore laser di una serie di barcode stampati su carta.

Questi test hanno dato esito positivo. In particolare il lettore laser, una volta connesso su una porta USB del computer con funzionamento in emulazione di tastiera, ha fornito i codici rappresentati senza errori, come è possibile verificare con il barcode di figura 3.

Esistono tuttavia altri tipi di verifiche, come l'efficacia di utilizzo della sintassi dei comandi e l'efficienza di esecuzione misurata nei tempi in secondi di compilazioni di sorgenti prestabiliti su una macchina di riferimento, nell'intento di valutare grandezze relative e confrontare implementazioni differenti della stessa libreria.

Per il primo aspetto presento il codice compilabile di un sorgente minimo d'esempio. Per farlo funzionare occorre salvare nel file di nome `lib-code39.lua` il codice del listato precedente, nella stessa cartella su disco del file sorgente di prova seguente:

```

% caricamento libreria
\directlua{
code39 = require("lib-code39.lua")
}

```



FIGURA 3: Il codice a barre con l'indirizzo del sito web dell'associazione $\mathcal{G}_I\mathcal{T}$, creato con il codice Lua_T_E_X illustrato nel testo.

Test Code 39

```

% barcode with default option
\directlua{
code = code39.validate("WWW.GUITEX.ORG")
code39.typeset(code)
}

% barcode with specified height and symbol
\directlua{
code39.typeset(code,{h="50pt",symb=true})
}

% local barcode object
\directlua{
local c = code39.validate("ABC123")
code39.typeset(c,{h='12pt',symb=true})
}
\bye

```

Per il secondo aspetto invece ho pensato di realizzare un documento contenente 10 000 barcode degli stessi dodici caratteri. A ogni esecuzione, la funzione di disegno genera da capo tutta la sequenza di nodi necessaria, perciò si tratta di una prova che sollecita in particolare la generazione dei nodi di Lua_T_E_X senza impegnare invece l'interruzione di riga. Il sorgente di prova è il seguente:

```

% loading library
\directlua{
c39 = require("lib-code39.lua")
}

\leavevmode\directlua{
local code = "ABC123456789"
local c = c39.validateCode39(code)
local opt = {h='12pt',symb=true}

% hugly
for i = 1,10000 do
    c39.typesetCode39(c, opt)
end
}
\bye

```

L'esecuzione è portata a termine in circa 4s con Lua_T_E_X 0.80, il ché vorrebbe dire circa 2500 barcode per secondo, ma è un dato assoluto che dovrebbe invece essere solamente confrontato con altri ottenuti sulla stessa macchina.

4 Conclusioni

Esplorare la libreria interna di Lua_T_E_X dedicata ai nodi è un ottimo modo per avvicinarsi a questo nuovo motore di composizione che promette di ampliare notevolmente le applicazioni da cui trarre vantaggio per comporre documenti di qualità.

Il linguaggio Lua da usare per programmare i nodi si dimostra un ottimo strumento perché con una sintassi essenziale offre un buon numero di concetti, compreso quello relativo alle funzioni di prima classe del paradigma funzionale, le closure,

il tipo di dati tabella molto generale, efficiente e con un costruttore dalla sintassi intuitiva davvero potente.

Internamente a LuaT_EX i nodi sono stati implementati come tipi *userdata* — cioè tipi previsti da Lua per estendere il linguaggio usando codice scritto in C — ma essi non sono gestiti in memoria nel modo usuale e automatico, ovvero con il Garbage Collector, un processo che elimina i dati non più utilizzabili in esecuzione. Questo perché non potremmo evitare che il Garbage Collector possa cancellare un nodo non ancora processato da T_EX, che opera in un ambito parallelo prelevando dati dalle proprie liste principali.

Nel programmare con i nodi e con le altre librerie dovremo quindi fare molta attenzione per gestire la memoria correttamente, pena il crash del processo. Altri pericoli vengono dagli errori sulla gestione della lista tipografica T_EX in attesa di processare gli elementi da noi prodotti con Lua. Spetta allo sviluppatore Lua mantenere in uno stato coerente le liste principali del compositore.

Nell'esempio applicativo abbiamo incontrato la necessità di *progettare* la struttura del codice valutando pro e contro delle alternative possibili e mostrando alcune tecniche utili con Lua. Le considerazioni d'efficienza d'esecuzione, d'interazione con le altri componenti software, di correttezza d'uso della memoria e la capacità del programma di sviluppo futuro del codice assumono importanza maggiore rispetto alla programmazione T_EX/L^AT_EX, richiedendo agli sviluppatori qualche dote in più nel campo dell'ingegneria del software proprio per coglierne tutte le potenzialità.

Rimane ancora molto da esplorare, e tutto lascia intuire applicazioni tipografiche molto difficili se non proprio impossibili da realizzare con altri software, anche se con LuaT_EX si dovrà procedere con cautela.

5 Ringraziamenti

È il momento finale dei ringraziamenti che vanno a Luigi Scarso, che mi ha aiutato — praticamente

in tempo reale — spiegandomi i dettagli spesso intricati dei nodi di LuaT_EX e durante tutto lo sviluppo di questo articolo introduttivo. Tuttavia, gli errori che comunque spero di non aver commesso, sono da imputare solamente a me.

Ringrazio Claudio Beccari per le sue informazioni T_EXniche su quando il compositore passa dal modo verticale a quello orizzontale — colmando un'evidente mia lacuna — e su alcune finzze del font design.

Grazie al team di LuaT_EX per avermi concesso l'uso del logo del progetto su interessamento di Luigi Scarso — sempre in tempo reale ovviamente.

Ringrazio infine la redazione di ArsT_EXnica per l'accurato lavoro di revisione e il lettore che ha svolto con passione gli *esercizi* proposti nel testo.

Riferimenti bibliografici

- GIACOMELLI, R. e PIGNALBERI, G. (2015). «Generare documenti L^AT_EX con diversi linguaggi di programmazione». *ArsT_EXnica*, (20), pp. 40–73. URL <http://www.guitex.org/home/it/numero-20-ottobre-2015>.
- Haskell (2016). «Official web site». <https://www.haskell.org/>.
- IERUSALIMSKY, R. (2013). *Programming in Lua, Third Edition*. Lua.Org.
- Rust (2016). «Official web site». <https://www.rust-lang.org/>.
- THE L^AT_EX DEVELOPMENT TEAM (2016). *LuaT_EX Reference Manual*, 0.80.0 edizione.
- Wikipedia (2016). «Code 39». https://en.wikipedia.org/wiki/Code_39.

▷ Roberto Giacomelli
Carrara
giaconet dot mailbox at gmail
dot com

La gestione delle lingue e delle loro varianti

Claudio Beccari

Sommario

Le varianti linguistiche presentano alcuni problemi che l'utente normale del sistema $\text{\TeX}$ non vede, ma che chi scrive classi o moduli per `babel` o per `polyglossia` deve affrontare e risolvere.

Questo articolo è in parte una introduzione alla gestione delle lingue, e in parte un tutorial su come risolvere alcuni problemi.

Abstract

Linguistic variants set forth some problems that the normal $\text{\TeX}$ system user does not perceive; but class, package, and extension module writers for `babel` and `polyglossia` must face and solve.

This paper in part introduces the reader to the ways the $\text{\TeX}$ system manages several languages and in part describes some methods to solve the above mentioned problems.

1 Introduzione

Usando $\text{\LaTeX}$ con il programma `pdf $\text{\LaTeX}$` o con i programmi `x $\text{\LaTeX}$` e `l $\text{\LaTeX}$` siamo ormai abituati a richiamare i pacchetti `babel` o `polyglossia` specificando loro le lingue da usare; raramente ci chiediamo che cosa ci sia dietro. Spesso non siamo nemmeno consci che molte lingue hanno delle varianti grandi o piccole, ma che richiedono di specificarle con nomi diversi, o con modificatori o attributi diversi, o con opzioni diverse.

1.1 Selezione delle lingue con `babel`

La tipica sintassi di `babel` per specificare le lingue da usare in un dato documento è la seguente:

```
\usepackage[⟨lista di lingue⟩]{babel}
```

ove la *⟨lista di lingue⟩* contiene una lista di nomi di lingue separati da virgole. Ogni lingua può essere accompagnata da uno o più modificatori specificati aggiungendo dopo il nome di una lingua il nome del modificatore preceduto da un punto; per esempio `latin.classic`. La lingua principale del documento può essere specificata premettendole la chiave `main=`, oppure specificandola come ultima nella *⟨lista delle lingue⟩*. Ci sono altri modi per impostare le lingue, ma questo è quello normalmente usato. L'utente si accorge immediatamente che il modificatore `.classic` nell'esempio precedente specifica una varietà di latino diversa da quella che si avrebbe non specificando nulla. Ma in che cosa differiscono? E queste differenze come sono descritte?

1.2 Selezione delle lingue con `polyglossia`

Per selezionare le lingue con `polyglossia` le cose sono leggermente diverse nella forma, sostanzialmente diverse nella sostanza. La sintassi generale è la seguente:

```
\usepackage{polyglossia}
\setmainlanguage[⟨opzioni⟩]{⟨lingua principale⟩}
\setotherlanguage[⟨opzioni⟩]{⟨lingua secondaria⟩}
\setotherlanguages{⟨lista di lingue secondarie⟩}
```

Come si vede, infatti la *⟨lingua principale⟩* è specificata con le sue *⟨opzioni⟩*; poi si specificano le lingue secondarie; quelle che richiedono *⟨opzioni⟩* vanno specificate una per una, ciascuna con le sue *⟨opzioni⟩*; quelle che non richiedono opzioni possono venire specificate con un unico comando mediante una *⟨lista di lingue secondarie⟩* costituita da una lista di nomi separati da virgole.

Sembra un poco più complicato rispetto a `babel`, ma è solo forma; la sostanza è che i modificatori di `babel`, una volta impostati non possono più venire cambiati, mentre le specificazioni di ciascuna lingua con `polyglossia` possono essere ripetute con opzioni diverse in diversi punti del documento; questa è una sostanziale differenza.

1.3 I font per le lingue

Con `pdf $\text{\LaTeX}$` si possono usare solo font codificati con non più di 8 bit; i font storici del sistema $\text{\TeX}$, i Computer Modern, sono codificati solo con 7 bit, il che significa che si possono usare solo 128 caratteri; gli altri font, generalmente, sono codificati con 8 bit e quindi si possono usare 256 caratteri. È evidente che i font storici non dovrebbero mai venire usati per comporre in lingue diverse dall'inglese che, fra le lingue moderne, sembra l'unica a non fare uso di segni diacritici.

Comunque il limite dei 256 caratteri, appena sufficiente per scrivere in caratteri latini estesi, non copre tutte le esigenze delle varie lingue europee scritte in caratteri latini; figuriamoci se il documento dovesse contenere brani di testo da comporre con i caratteri greci oppure cirillici. Cambiare lingua significa quindi dover cambiare anche set di caratteri, che ovviamente non sono codificati come quelli latini; il cambio di lingua, quindi, coinvolge molti aspetti diversi che non sono costituiti solo da un diverso lessico.

Con `x $\text{\LaTeX}$` che può gestire i font OpenType le cose sono un poco più semplici. Anche `l $\text{\LaTeX}$` può lavorare con i font OpenType, ma la sua potenza richiede di gestire i font in modo leggermente diverso da come li si gestisce con `x $\text{\LaTeX}$` , perciò nel

seguito ci si soffermerà solo sull'uso del programma *xelatex*.

I font OpenType sono delle collezioni enormi di caratteri; sono generalmente codificati in UNICODE e vi si accede anche con la transcodifica UTF-8. Senza scendere nei dettagli, i font OpenType possono contenere decine di migliaia di caratteri appartenenti agli alfabeti più diversi; possono contenere segni analfabetici, pittogrammi, ideogrammi, eccetera. Nessun font OpenType contiene tutti i caratteri accessibili con la codifica UNICODE, ma l'enorme sproporzione fra i 256 caratteri massimi dei font gestibili con *pdflatex* e le molte decine di migliaia di caratteri dei font OpenType gestibili con *xelatex* balza agli occhi. Con *xelatex* cambiare lingua non implica generalmente l'obbligo di cambiare font.

Per esempio, un font libero come il Linux Libertine O contiene i caratteri latini con ogni possibile estensione; i caratteri greci con ogni variante grafica; i caratteri ebraici; i caratteri cirillici estesi per comporre nel centinaio di lingue che usano quell'alfabeto; contengono anche i caratteri dell'*alfabeto fonetico internazionale*; in totale contiene 'solo' 2674 segni. Gli Hiragino Mincho ProN, ne contengono 20 325 e non sono completi, perché pare che gli ideogrammi usabili in cinese e in giapponese siano circa 35 000.

È quindi chiaro che quando si devono usare alfabeti diversi è molto conveniente servirsi del programma *xelatex* o di *lualatex*¹

Vale la pena di ricordare che con *polyglossia* è possibile specificare con `\newfontfamily` il nome di una famiglia di caratteri OpenType con caratteristiche speciali per comporre in una data lingua: basta che il nome della nuova famiglia sia della forma `\langle lingua \rangle font`. La nuova famiglia di font potrebbe contenere un alfabeto che la famiglia principale non contiene; oppure potrebbe essere la stessa famiglia del font principale, ma con caratteristiche diverse; per cui quando si usa l'ambiente `\langle lingua \rangle`, viene automaticamente selezionata la famiglia di font `\langle lingua \rangle font` senza bisogno di impostare nulla. Purtroppo una funzionalità così comoda non è disponibile per quando si usa *babel*; tuttavia non è difficile definirsi ambienti che di fatto realizzino le stesse funzionalità.

1. Se non si devono usare la funzionalità estese con il linguaggio Lua incorporato in *lualatex*, sembrerebbe equivalente usare i font OpenType con l'uno o l'altro programma; vi è una sottile ma importante differenza; *xelatex* può usare la microtipografia solo per quanto riguarda la protrusione di piccole parti dei caratteri nei margini; invece *lualatex* può usare anche l'espansione dei caratteri; con questa tecnica la giustificazione delle righe di un capoverso non è limitata allo sfruttamento dello spazio interparola, ma anche può ricorrere alla dilatazione o contrazione orizzontale dei caratteri, entrambe limitate a percentuali piccolissime, impercettibili a occhio nudo, ma sufficienti a compensare visibilmente lo spazio interparola.

2 La gestione delle lingue

Quanto esposto nell'Introduzione è del tutto evidente, e ogni utente non ha in realtà nessun bisogno che gli si ricordi come stanno le cose.

Ricapitoliamo però come funziona quello che usiamo quotidianamente.

Per gestire una *\langle lingua \rangle*, *babel* e *polyglossia* hanno bisogno essenzialmente di un file chiamato rispettivamente *\langle lingua \rangle .ldf* e *gloss-\langle lingua \rangle .ldf*. Questi due file contengono in principio le stesse informazioni, ma sono scritte con sintassi diverse.

1. Viene eseguita la verifica che esista la definizione dei pattern di sillabazione per la *\langle lingua \rangle*. Se non esistesse un insieme di pattern associati alla *\langle lingua \rangle*, *babel* la definisce come un 'dialetto' dell'inglese americano; siccome la sillabazione di questa lingua è molto particolare, le parole che cadessero in fin di riga verrebbero sillabate, sì, ma in un modo che in generale è ridicolo. Invece *polyglossia* associa alla *\langle lingua \rangle* l'insieme di pattern vuoto, associato al nome di 'lingua' *nohyphenation*² e nessuna parola della lingua viene divisa in sillabe in fin di riga.

Se i pattern esistono l'impostazione della *\langle lingua \rangle* seleziona anche i pattern giusti. Quando si dice che "i pattern esistono" si intende che i relativi file sono stati caricati nel file di formato *pdflatex.fmt* o *xelatex.fmt*. Invece *lualatex* contiene già caricati i pattern solo dell'inglese americano e carica al volo i pattern delle altre lingue specificate a *polyglossia*; questo ritarda la partenza della composizione vera e propria, ma risparmia non poca memoria, usabile proficuamente per altri scopi.

2. Viene anche eseguita la verifica che il file *.ldf* non sia già stato caricato, per evitare di ridefinire certe macro, magari in modo diverso e non corretto.
3. Vengono definiti i comandi `\captions\langle lingua \rangle` che contengono tutte le parole fisse come 'Capitolo', 'Figura', eccetera, usate nelle intestazioni dei capitoli, nelle didascalie, e in molte altre situazioni dove bisogna usare delle parole fisse in locuzioni costruite dal programma di composizione.
4. Viene definito il modo di scrivere la data corrente mediante il comando `\today`.
5. Vengono impostati i parametri di composizione, fra cui i rientri dei capoversi, le penalità, le lunghezze minime del primo e dell'ultimo frammento di parola, cioè la stringa prima del primo punto di cesura o dopo l'ultimo punto

2. La pseudolingua *nohyphenation* in realtà è associata ad un file di pattern che ne contiene solo due; esiste un altro set di pattern che è veramente vuoto. Vengono usati, o l'uno o l'altro, per disattivare la divisione in sillabe.

di cesura. L'esempio minimo per l'italiano è formato dalla parola 'idea' che secondo la grammatica si potrebbe dividere in sillabe come 'i-de-a'; senonché per l'italiano le lunghezze minime del primo e dell'ultimo frammento di parola sono fissate entrambe di due caratteri, quindi la parola non si può dividere affatto. Questo fatto, tra l'alto, mostra che la cesura tipografica è diversa dalla cesura grammaticale; questa differenza esiste per tutte le lingue.

6. Tutte queste impostazioni vengono accumulate dentro una macro che si chiama `\extras⟨lingua⟩` in modo che ogni qual volta si seleziona la composizione in quella lingua, viene eseguita la macro e quindi tutte le impostazioni vengono ristabilite.
7. Tutte le impostazioni vengono annullate con comandi che vengono accumulati dentro la macro `\noextras⟨lingua⟩`, in modo che quando si finisce di comporre in quella lingua, viene azzerata ogni impostazione specifica per consentire di ricominciare la composizione in un'altra lingua senza residui di quella precedente.
8. Molto spesso le lingue hanno bisogno di comandi particolari che per la comodità di inserimento del testo dovrebbero essere i più semplici e meno invasivi possibile. A questo scopo si definiscono uno o più caratteri attivi per far loro svolgere determinati compiti. Un carattere attivo è un normale carattere ASCII che viene attribuito alla categoria 13; questi caratteri possono ricevere una definizione in modo analogo a quello che succede con i normali comandi che cominciano con la barra rovescia. Per esempio si usa sovente il carattere attivo `~`, predefinito nel nucleo di L^AT_EX, a cui è stata data la definizione `\nobreakspace`; esso infatti lega due parole con uno spazio indivisibile.

Il carattere usato più di frequente per le lingue è `"`, ma ne possono venire definiti altri; il modulo per il francese di `babel` definisce attivi tutti i segni di interpunzione esclusi la virgola e il punto fermo, perché, a differenza di quasi tutte le altre lingue, il francese prevede uno spazio indivisibile prima di ognuno di quei segni. In questo modo chi compone in francese batte normalmente, per esempio, il carattere del punto e virgola senza preoccuparsi di lasciare spazi nel file sorgente, perché ci pensa il punto e virgola attivo a togliere eventuali spazi che lo precedono per rimetterci lo spazio indivisibile e poi un punto e virgola 'normale'.

9. Anche l'attivazione e la disattivazione dei caratteri attivi vengono aggiunte alle macro `\extra...` e `\noextras...` in modo da attivare i caratteri attivi con le definizioni giuste

per ciascuna lingua e lasciare il campo 'pulito' quando si cessa di comporre in quella lingua.

10. Se con `babel` si specificano dei modificatori per una data lingua le loro definizioni particolari per cambiare modalità di composizione vengono definite e aggiunte a `\extras...` e le macro che disattivano queste varianti vengono aggiunte a `\noextras...`
11. Se le impostazioni delle lingue per `polyglossia` comprendono delle *⟨opzioni⟩*, queste vengono trattate in modo simile ai modificatori di `babel` salvo che se si specifica nuovamente una data lingua con *⟨opzioni⟩* diverse, queste sostituiscono le precedenti.

Come si vede dietro le quinte la specificazione delle lingue richiede un gran lavoro che coloro che scrivono i moduli per `babel` e per `polyglossia` devono collaudare con grande cura perché si tratta di impostazioni molto delicate.

3 La divisione in sillabe

Come si è detto, ogni lingua dovrebbe essere accompagnata da uno o più file di pattern di sillabazione; eventualmente anche da uno o più file con gli elenchi delle parole che "fanno eccezione" rispetto alle regole predisposte dai pattern.

Infatti quando uno qualsiasi dei programmi di composizione compone un capoverso fa un grosso lavoro per trasformare la lunga lista di caratteri, penalità, crenature, spazi rigidi ed elastici, scatole di vario tipo corrispondenti all'input eseguito e dalle macro usate dall'utente, in un capoverso diviso in righe di uguale lunghezza (tranne l'ultima); deve fare molte prove e molte verifiche per ottenere la composizione migliore. Non è il caso qui di scendere nei dettagli, ma merita mostrare i tre passaggi con cui il programma *paragraph builder* spezza le righe per formare un capoverso ben composto.

1. In prima battuta il programma cerca di spezzare la lunga lista in righe di uguale lunghezza senza attivare l'algoritmo di sillabazione; se la scomposizione avviene con un fattore di 'bruttezza' inferiore ad un certa tolleranza, accetta il risultato e il lavoro del *paragraph builder* finisce.
2. In seconda battuta spezza il capoverso in righe di uguale lunghezza usando la sillabazione; nel fare questo esamina anche la liste delle eccezioni in modo da non usare punti di cesura errati perché le parole che fanno eccezione verrebbero divise in modo errato se non fossero, appunto, elencate come eccezioni; esamina i pattern relativi alla lingua o alla variante della lingua e cerca la migliore divisione delle righe in modo che la 'bruttezza' sia minima e sia inferiore ad un altro diverso valore di tolleranza. Se ci riesce, bene, il capoverso è composto.

3. In terza battuta allarga o restringe gli spazi interparola più di quanto sia consentito, oppure rinuncia a restare entro i margini, cosicché una o più righe escono dai margini. In questi casi lascia il capoverso diviso male, ma informa l'utente che ci sono delle *underful hbox* (righe riempite con spazi interparola troppo grandi) o *overful hbox* (righe troppo piene che escono dal margine destro), affinché l'utente possa provvedere con una risistemazione del capoverso consistente in una riformulazione delle frasi, accorciandole, allungandole, scambiandole di posto.

Si capisce quindi che la divisione dei capoversi in righe giustificate richiede un ottimo algoritmo di sillabazione.

Certo ci sono alcune lingue che non hanno bisogno della cesura in fin di riga; per esempio quelle che si scrivono con ideogrammi non hanno bisogno di cesure, perché si può andare a capo fra un ideogramma e l'altro senza problemi.

Altro esempio: l'arabo si scrive con caratteri legati molto calligrafici; in questa lingua i capoversi vengono giustificati allungando il tratto finale delle lettere terminali quanto basta per allungare la riga fino al margine; questa coda allungabile delle lettere terminali si chiama *kashida* e viene usata moltissimo.

Alcune lingue non sono dotate di pattern propri; in questo modo vengono dichiarati dialetti di altre lingue con cui condividere la sillabazione; se chi scrive i file `.ldf` non si ricorda di questo fatto rischia di avere le parole della sua *lingua* divise con le regole per l'inglese americano o di non essere divise affatto perché la lingua diventa un 'dialetto' di *nohyphenation*.

4 I pattern di sillabazione

Per dividere in sillabe i vari programmi, come certi word processor e certi text processor, usano lunghi elenchi con le parole già divise in sillabe. Il metodo non è malvagio, ma richiede elenchi lunghissimi e un ingombro di memoria molto grande oltre ad un tempo di elaborazione lungo; inoltre questi elenchi non sono mai completi.

Altri programmi usano programmi che implementano le regole grammaticali di divisione in sillabe; per certe lingue questo metodo funziona abbastanza bene.

Altri programmi ancora usano i pattern di sillabazione; non si tratta solo dei programmi di composizione del sistema T_EX, ma anche dei word processor delle collezioni OpenOffice e LibreOffice, di InDesign e di diversi altri programmi.

I pattern sono frammenti di parola formati da una sola lettera, due lettere, tre lettere, eccetera, dove ogni lettera è affiancata da entrambi i lati da una cifra fra 0 a 9, estremi compresi; lo zero è

sottinteso e quindi spesso non viene esplicitato ma c'è sempre anche se in modo non esplicito. Le cifre pari indicano il divieto di dividere con maggiore o minore forza a seconda del valore più o meno grande della cifra; le cifre dispari indicano il consenso di dividere con maggiore o minore forza a seconda del valore numerico. Se tra le stesse due lettere presenti in pattern diversi compaiono cifre diverse, prevale quella di valore maggiore. Quindi smembrando la parola da dividere in stringhe di una sola lettera, di due lettere, eccetera, e cercando nell'elenco dei pattern quali cifre ci sono fra quali lettere si arriva rapidamente ad esaurire e a completare la parola con le opportune cifre pari o dispari fra ogni coppia di lettere. Perché l'algoritmo funzioni rapidamente bisogna che i pattern siano conservati in memoria in una struttura di dati che ne consenta l'esplorazione più rapida possibile. I programmi di composizione del sistema T_EX sono attrezzati per questo scopo e svolgono questo lavoro con grandissima rapidità.

Va da sé che l'interpretazione dei pattern come sequenze di stringhe alfanumeriche è piuttosto semplice. La parte difficile è costruire questi pattern.

Per l'inglese l'operazione è molto difficile perché la divisione delle parole dipende dalla pronuncia e non dalla grafia; si pensi che il nome *record* e il verbo *record* si pronunciano in modo diverso e si dividono rispettivamente in *rec-ord* e *re-cord*; lo stesso accade per il nome *analyses* e il verbo *analyses* che si dividono rispettivamente in *anal-yses* e *ana-lyses*. È evidente che senza una analisi grammaticale associata alla sillabazione è impossibile dividere correttamente queste parole omografe.

Tuttavia il dottorando Frank Liang, che lavorava con Donald Knuth nelle fasi iniziali della creazione del sistema T_EX, produsse un programma *patgen* per creare i pattern a partire da una lista di 25 000 parole inglesi già divise in sillabe; vista la fattibilità e l'efficacia del programma, lo si applicò ad una lista di 100 000 parole, e con questo si produsse il primo file `hyphen.tex` ancora oggi presente (con un nome leggermente cambiato) in ogni distribuzione del sistema T_EX; conteneva quasi 5000 pattern e divideva correttamente il 90% delle parole inglesi. I pattern per l'italiano contengono circa 340 pattern e dividono correttamente il 100% delle parole che non derivino da radici straniere. Questo la dice molto lunga sulla difficoltà di dividere in sillabe certe lingue. I pattern per l'italiano sono stati costruiti a mano partendo dalla grammatica e aggiungendo un po' di conoscenza della lingua per affrontare la divisione di parole entrate nell'uso italiano ma straniere o assimilate; vengono divise correttamente parole come *leishmaniosi*, *massmediologo*, *newyorkese*, *maxwelliano*, *taylorismo*, ed altre simili.

Una volta creati i pattern di una lingua bisogna corredarli di altri due file; uno per caricarli

durante al creazione del formato e l'altro per elencare le debite corrispondenze di certi segni usati normalmente per scrivere i file sorgente `.tex` e i corrispondenti simboli con la codifica UNICODE; un segno tipico è l'apostrofo, oltre, evidentemente, i caratteri nazionali con diacritici.

Tanto per stare in tema, il latino ha quattro ortografie, quindi necessita di quattro varianti o di quattro modificatori:

- il latino moderno che usa la u e la v e non usa i dittonghi legati;
- il latino medievale che usa solo la u e la V e i dittonghi legati;
- il latino classico che usa solo la u e la V e non usa i dittonghi legati;
- il latino ecclesiastico che usa lo stesso alfabeto del latino moderno, ma usa gli accenti tonici, i dittonghi legati anche questi eventualmente dotati di accento; inoltre usa le note al piede come in francese e una spaziatura dei segni di interpunzione alti, sia pure con spazi meno ampi del francese.

Il latino vuole due file di pattern; uno, `hyph-la.tex`, gestisce tutte le varianti tranne quella classica; un secondo, `hyph-la-x-classic.tex`, gestisce la sillabazione classica. I due tipi di sillabazione differiscono non solo per l'ortografia, ma anche perché le versioni moderne e medievale sono divisioni fonetiche, mentre la divisione classica è etimologica, molto più difficile da programmare della divisione fonetica.

Quindi lo stesso file di gestione del latino, oltre a comprendere le impostazioni dei modificatori o delle varianti deve anche saper associare il numero giusto che identifica i set di pattern a seconda che si usi la sillabazione fonetica o quella etimologica.

Il modulo `latin.ldf` per `babel` definisce l'attributo (o modificatore) per il latino classico con i seguenti comandi:

```
\bbl@declare@attribute{latin}{classic}{%
\expandafter\addto\expandafter
\extraslatin\expandafter{\extrasclassic}%
}
\ifx\l@classicalatin\undefined
\let\l@classicalatin\l@latin
\PackageWarningNoLine{babel}{%
Attention: hyphenation patterns
for language\MessageBreak
classicalatin have not been
loaded.\MessageBreak
I go on using the modern Latin hyphenation
patterns.\MessageBreak
Please, load the suitable patterns or
upgrade your TeX distribution}
\fi
\addto\extrasclassic{%
\let\l@latin\l@classicalatin}
```

L'attributo viene definito aggiungendo le impostazioni di `\extrasclassicalatin` a quelle di

`\extraslatin`, ma poi verifica che la lingua `\l@classicalatin` sia stata definita nel formato quando si sono caricati i pattern. Non è detto che lo siano, perché l'installazione potrebbe essere obsoleta o più spesso parziale; in questo caso il nome della lingua `\l@classicalatin` viene reso equivalente a quello del latino moderno; un avviso viene stampato nel file `.log` per informazione dell'utente, ma non viene impostato nessun flag d'errore; ovviamente la composizione in latino classico non avrà le cesure corrette, ma almeno la composizione andrà a termine. In ogni caso il nome della lingua `\l@latin` viene reso equivalente a quello della lingua `\l@classicalatin` e questa equivalenza diventa permanente perché viene anche aggiunta alla lista delle impostazioni contenuta in `\extrasclassicalatin`.

Questo procedimento sembra molto contorto, ma in realtà funziona benissimo con `babel` dove le impostazioni sono sempre permanenti una volta che sia attivata la lingua. Con `polyglossia` le cose sono un po' più semplici, perché le impostazioni per una data lingua possono essere cambiate in qualunque momento.

Per il greco la situazione è ancora più complessa. La necessità con `pdflatex` di cambiare alfabeto, e perciò cambiare codifica dei font in uscita, implica molte azioni in più. Inoltre il greco presenta tre varianti ortografiche e lessicali:

- il greco moderno monotonico, che usa solo il 'tonos', generalmente rappresentato con un accento acuto, ma non usa tutta la varietà di accenti delle altre varianti; conserva solo la dieresì;
- il greco moderno politonico che usa tutti i diacritici: gli spiriti e la dieresì; i tre accenti grave, acuto e circonflesso; lo iota sottoscritto o adscritto;
- il greco antico che usa tutta la serie dei diacritici usati dal greco moderno politonico; il lessico è decisamente diverso; la divisione in sillabe è ovviamente diversa.

Il fatto che i segni diacritici si spostino su altre sillabe a seconda dei suffissi della declinazione dei nomi e degli aggettivi e della coniugazione dei verbi, fa sì che i pattern siano piuttosto complicati.

A tutto questo bisogna aggiungere che con `pdflatex` si possono usare solo font codificati con non più di 8 bit, mentre con `xelatex` i font OpenType consentono di codificare i font con un numero elevato di bit, fa sì che i pattern per il greco da comporre con `pdflatex` siano diversi da quelli per il greco da comporre con `xelatex`; non sono diversi come regole perché comunque sia composta, ciascuna parola deve essere sillabata nello stesso modo; sono diversi perché i caratteri da indicare nei file di pattern per `pdflatex` sono necessariamente codificati in modo diverso da quelli usati per i file di pattern per `xelatex`.

In conclusione la gestione del greco richiede sei file di pattern, tre per ciascuna variante da comporre con *pdflatex* e tre per quelle da comporre con *xelatex*.

Considerazioni simili si possono fare per tutte le lingue che presentano delle varianti.

5 La gestione delle varianti

Come si è detto le varianti sono specificate mediante $\langle \text{modificatori} \rangle$ con *babel* e con i $\langle \text{valori} \rangle$ nelle opzioni della forma `variant =  $\langle \text{valore} \rangle$` con *polyglossia*.

Per l'utente non c'è altro da fare se non ricordare che l'impostazione dei $\langle \text{modificatori} \rangle$ è permanente per l'intero lavoro di composizione; invece i $\langle \text{valori} \rangle$ specificati nelle opzioni per l'impostazione delle lingue con *polyglossia* possono venire cambiati ripetutamente, anche se non sembra una cosa particolarmente elegante in tipografia, sebbene talvolta possa essere necessaria.

Dietro le quinte *babel* gestisce le impostazioni mediante le liste contenute dentro i comandi `\extras...` e `\noextras...`; a loro volta queste specifiche impostazioni per le varianti possono modificare determinate impostazioni o possono coinvolgere variabili booleane per fare certe cose a seconda del modificatore impostato.

Per *polyglossia* le cose funzionano in modo analogo specialmente mediante la gestione delle opzioni con la sintassi specificata dal pacchetto *xkeyval* che, oltre ad usare la sintassi $\langle \text{chiave} \rangle = \langle \text{valore} \rangle$, non solo distingue le opzioni booleane da quelle che vogliono dei valori diversi, ma controlla anche che questi diversi valori appartengano ad una lista di valori accettabili; questi valori possono essere delle 'parole' o dei numeri o delle dimensioni, non importa, ma la sintassi di *xkeyval* è molto precisa nel distinguerle e a generare messaggi adeguati nel caso che vengano specificati valori non ammissibili.

Tuttavia ogni variante, sia con *babel* sia con *polyglossia*, deve venire programmata in modo corretto e specifico per ogni possibile situazione.

Nel seguito, senza scendere nei dettagli troppo fini, si mostrerà, per esempio, come vengono gestiti gli spazi che precedono la punteggiatura in francese e nella variante ecclesiastic del latino. Questo esempio serve per mostrare come risolvere un problema specifico per ottenere gli stessi risultati quando si usino font codificati a 8 bit, oppure font OpenType.

6 Gli spazi della punteggiatura

6.1 Con babel

Il problema degli spazi prima della punteggiatura si risolve ricorrendo a caratteri attivi, che con *babel*, non a caso, vengono chiamati *shorthands* (scorciatoie).

Le impostazioni per il francese sono contenute nel file `frenchb.ldf` che prevede di funzionare correttamente anche quando si usa *babel* con *xelatex* e *lua-latex*; in questi casi usa più o meno le stesse impostazioni di *polyglossia*. Ma con *pdflatex* bisogna ricorrere ai caratteri attivi. Il codice usato è il seguente:

```

1 \initiate@active@char{:}%
2 \initiate@active@char{;}%
3 \initiate@active@char{!}%
4 \initiate@active@char{?}%
5 \declare@shorthand{french}{:}{;}%
6 \ifhmode
7 \ifdim\lastskip>\z@
8 \unskip\penalty\@M\FBthinspace
9 \else
10 \FDP@thinspace
11 \fi
12 \fi
13 \string;}
14 \declare@shorthand{french}{!}{;}%
15 \ifhmode
16 \ifdim\lastskip>\z@
17 \unskip\penalty\@M\FBthinspace
18 \else
19 \FDP@thinspace
20 \fi
21 \fi
22 \string!}
23 \declare@shorthand{french}{?}{;}%
24 \ifhmode
25 \ifdim\lastskip>\z@
26 \unskip\penalty\@M\FBthinspace
27 \else
28 \FDP@thinspace
29 \fi
30 \fi
31 \string?}
32 \declare@shorthand{french}{:}{;}%
33 \ifhmode
34 \ifdim\lastskip>\z@
35 \unskip\penalty\@M\FBcolonspace
36 \else
37 \FDP@colonspace
38 \fi
39 \fi
40 \string:}
41 \declare@shorthand{system}{:}{\string:}
42 \declare@shorthand{system}{!}{\string!}
43 \declare@shorthand{system}{?}{\string?}
44 \declare@shorthand{system}{:}{\string;}
45 \FB@addto{extras}{%
46 \languageshorthands{french}%
47 \bbl@activate{:}\bbl@activate{;}%
48 \bbl@activate{!}\bbl@activate{?}%
49 }
50 \FB@addto{noextras}{%
51 \bbl@deactivate{:}\bbl@deactivate{;}%
52 \bbl@deactivate{!}\bbl@deactivate{?}%
53 }

```

```

1 \ifluatex
2   \newluatexattribute\xpg@frpt %
3   \directlua{polyglossia.load_frpt()}%
4 \else
5   \newXeTeXintercharclass\french@punctthin % ! ? ; et autres
6   \newXeTeXintercharclass\french@punctthick % :
7   \newXeTeXintercharclass\french@punctguillstart % « <
8   \newXeTeXintercharclass\french@punctguillend % » >
9 \fi
10
11 \def\xpg@unskip{\ifhmode\ifdim\lastskip>z@\unskip\fi\fi}
12 \def\xpg@nospace#1{#1}
13
14 \def\french@punctuation{%
15   \lccode"2019="2019
16   \ifluatex
17     \global\xpg@frpt=1\relax %
18     \directlua{polyglossia.activate_frpt()}%
19   \else
20     \XeTeXinterchartokenstate=1
21     \XeTeXcharclass ‘! \french@punctthin
22     \XeTeXcharclass ‘? \french@punctthin
23     \XeTeXcharclass ‘; \french@punctthin
24     \XeTeXcharclass ‘: \french@punctthick
25     % ...
26     \XeTeXcharclass ‘« \french@punctguillstart
27     \XeTeXcharclass ‘» \french@punctguillend
28     \XeTeXcharclass ‘< \french@punctguillstart
29     \XeTeXcharclass ‘> \french@punctguillend
30     \XeTeXinterchartoks\z@ \french@punctthin={\nobreak\thinspace}%
31     \XeTeXinterchartoks\z@ \french@punctthick={\nobreakspace}%
32     \XeTeXinterchartoks255 \french@punctthin={\xpg@unskip\nobreak\thinspace}%
33     \XeTeXinterchartoks255 \french@punctthick={\xpg@unskip\nobreakspace}%
34     \XeTeXinterchartoks\french@punctguillstart\z@={\nobreakspace}% "«" -> "« a"
35     \XeTeXinterchartoks\z@ \french@punctguillend={\nobreakspace}% "a»" -> "a »"
36     \XeTeXinterchartoks\french@punctguillstart255={\nobreakspace\xpg@nospace}% "« " -> "«~"
37     \XeTeXinterchartoks255 \french@punctguillend={\xpg@unskip\nobreakspace}% " " »" -> "~»"
38     \XeTeXinterchartoks\french@punctguillend\french@punctthin={\nobreak\thinspace}% "»;" -> "» ;"
39     \XeTeXinterchartoks\french@punctguillend\french@punctthick={\nobreakspace}% "»:" -> "» :"
40     \XeTeXinterchartoks\french@punctthin\french@punctguillend ={\nobreakspace}% "?»" -> "? »"
41   \fi
42 }

```

Nota. Dall'elenco dei segni da attribuire alla classe `\french@punctthin` ho tolto i segni ‘!’, ‘!?’ , ‘?!’ e ‘??’ che non sono disponibili nei font di default Latin Modern. D'altra parte questi glifi sono assenti anche dalla maggior parte dei font OpenType.

Come si vede vengono ripetute con piccole modifiche le stesse definizioni per tutti e quattro i segni di interpunzione alti.

Dapprima con `\initiate@active@char` si specifica che il carattere specificato come argomento deve essere assegnato alla categoria dei caratteri attivi; con il comando `\declare@shorthand` si assegna una definizione al carattere attivo del secondo argomento e lo si vincola alla lingua specificata nel primo argomento; questo consente di aggiungere la definizione alla lista `\languageshorthands`, come si vede nella riga 46, per potere aggiungere questa lista e l'attivazione dei caratteri attivi agli `\extras...` del francese.

Per ciascun carattere si danno definizioni diverse solo se esso è usato in modo orizzontale (test con `\ifhmode`); poi si verifica se prima di agire è stato inserito dello spazio e nel caso lo si elimina mediante `\unskip` poi si inserisce una penalità 'infinita'³ (`\penalty\@M`) prima di inserire lo spazio specifico richiesto dal particolare segno di interpunzione; infine viene inserito il segno di interpunzione 'normale' ottenendo premettendo il comando `\string` allo stesso segno attivo di cui si sta dando la definizione. Il comando nativo `\string` trasforma in uno o più caratteri di categoria 12 (carattere diverso da una lettera) il token semplice (un solo carattere) o complesso (il nome di una macro compresa la barra rovescia) che lo segue.

Per essere sicuri di non combinare partecipi quando si cambia lingua non solo si aggiungono i comandi di "disattivazione" alla lista contenuta nei `\noextras...` (righe 50–53), ma nelle righe 41–44 ci si assicura che i segni di interpunzione alti siano non attivi a livello di sistema, o meglio, in modo che, se il sistema T_EX ha bisogno di usarli per suo conto, questi caratteri inseriscano gli stessi segni resi di categoria 12 mediante il comando `\string`.

I vari spazi `\FBthinspace` e `\FBcolonspace` sono stati definiti prima nel file `frenchb.ldf`, e hanno la caratteristica di essere degli spazi elastici dotati di una certa allungabilità e di una certa accorciabilità. Per noi italiani che non siamo abituati a vedere spazi prima dei segni di interpunzione quegli spazi sembrano piuttosto larghi, tanto più se per motivi di giustificazione il *paragraph builder* allarga tutti gli spazi presenti nelle righe.

Infatti nelle definizioni analoghe per il latino ecclesiastico (pacchetto `ecclesiastic` richiamato da `babel` quando si specifica il modificatore `ecclesiastic`), gli spazi sono definiti con valori minori, ma essenzialmente sono privi di allungabilità e di accorciabilità.

Tutto ciò è delicato, ma tutto sommato è piuttosto semplice.

3. Per i programmi del sistema T_EX una penalità di 10 000 è considerata infinita; la macro `\@M` indica il valore 10 000.

6.2 Con polyglossia

Con `polyglossia` le cose sono relativamente più semplici e non richiedono l'uso di caratteri attivi. Infatti i comandi per gestire i font OpenType permettono di definire delle classi di caratteri da popolare come si vuole, visto che inizialmente tutti appartengono alla classe 0 tranne lo spazio che appartiene alla classe 255. Quindi per il francese e per il latino ecclesiastico basta definire le classi dei segni di interpunzione alti e basta specificare quali spazi mettere fra quali caratteri di quali classi. Sembra oscuro ma è semplicissimo; rivediamo la soluzione per il francese; quella per il latino ecclesiastico è simile, ma più semplice perché serve definire una sola classe e un solo spazio. Il file `gloss-french.ldf` contiene, oltre altro codice, quello mostrato nella pagina 7

In questo codice si vedono dei comandi non molto consueti quando si scrivono macro; si tratta di comandi nativi di `xetex` che sfruttano le particolarità dei font OpenType.

Nel codice si vede all'inizio il test `\iflua` che controlla se il motore di composizione è `luatex`; se lo è, definisce un nuovo attributo di Lua e poi importa uno specifico pacchetto adatto per sfruttare le maggiori potenzialità di `luatex` rispetto a `xetex`. Altrimenti definisce quattro nuove classi di glifi: una per i punti di interpunzione alti che richiedono un piccolo spazio; una per i due punti che richiedono uno spazio maggiore; una per i caporali aperti che richiedono uno spazio a destra e un'altra per i caporali chiusi.

Dopo aver definito una macro per togliere eventuali spazi orizzontali eventualmente inseriti dall'utente, definisce le regole della "French punctuation" mediante la macro `\french@punctuation`. Leggendo questa macro vediamo che oltre ad accettare l'apostrofo UNICODE ("2019) fra le lettere accettabili dal francese per formare le parole⁴ e dopo aver separato le impostazioni se si sta lavorando con il motore di composizione `luatex`, assegna i vari segni di punteggiatura alle rispettive classi.

Poi vengono i comandi di spaziatura da usare fra i caratteri di diverse classi; il comando che fa tutto questo si chiama `\XeTeXinterchartoks` e lavora con la sintassi seguente:

```
\XeTeXinterchartoks <classe a sinistra> <classe a destra> = {<comandi>}
```

Ricordiamo: la classe 'zero' (indicata con `\z@`) rappresenta gli altri caratteri non appartenenti alle classi definite poche righe prima; la classe 255 in sostanza rappresenta la fine o l'inizio di una stringa di caratteri di classe 'zero'; le altre classi hanno i nomi definiti sopra. I `<comandi>` sono i comandi di spaziatura da inserire fra i due caratteri di sinistra

4. Questa impostazione in realtà non ha nulla a che vedere con la punteggiatura, ma va sempre applicata sia in francese sia nelle altre numerose lingue che ne fanno uso.

e di destra delle due classi elencate come primi due argomenti del comando.

Leggiamo quindi che fra un carattere normale seguito da un carattere che vuole la spaziatura sottile bisogna inserire due comandi: uno per impedire un fine riga e il secondo per inserire uno spazio sottile; due righe dopo leggiamo che fra la fine di una stringa di caratteri normali (indicata dal valore speciale 255) e un segno che richiede la spaziatura sottile bisogna inserire tre comandi: il primo per togliere eventuali spazi inseriti dall'utente, poi la penalità per evitare un fine riga, poi lo spazio fine.

Non affermo che sia semplicissimo, ma non è assolutamente complicato; se esiste qualche difficoltà, questa resta sulle spalle di chi scrive il file `gloss-⟨lingua⟩.ldf` perché deve curare ogni dettaglio e deve collaudare con attenzione tutte le possibili combinazioni di classi per essere sicuro di non aver scambiato o dimenticato qualche cosa.

7 Conclusioni

Questo articolo potrebbe diventare lunghissimo se volessi scrivere tutto quello che serve per gestire le lingue con `babel` e con `polyglossia`. Ma mi sono limitato all'essenziale, forse troppo stringato e quindi troppo superficiale.

In realtà le cose da fare per gestire le lingue e le loro varianti non sono molte e chiunque può cimentarsi. Se definisce i file di descrizione per una lingua nuova deve seguire queste regole semplicissime:

- deve scrivere il file `.dtx` che contiene il codice commentato del file `.ldf` in modo che produca la documentazione e il modo d'uso oltre che il codice commentato;
- deve predisporre un file `README` che contenga un minimo di informazioni sul file, dove e come installarlo e a quale licenza è soggetto;
- deve predisporre un file compresso in formato `.zip` contenente i file `.dtx`, `.pdf`, `README`, oltre all'eventuale file `.ins` se il file `.dtx` non è autoestraente;
- deve eseguire l'upload su CTAN con le modalità descritte nella pagina web <http://www.ctan.org/upload> dopo aver letto con attenzione quanto scritto in <http://tug.org/texlive/pkgcontrib.html>.

Se ha predisposto i file di pattern deve chiamarli con la sigla `hyph-⟨sigla ISO⟩.tex`⁵ inviandoli al gruppo di lavoro all'indirizzo di posta elettronica `tex-hyphen@tug.org`; il gruppo risponderà per indicare quali altri file inviare per gestire correttamente i caratteri o i segni alfabetici che non sono strettamente ASCII. Probabilmente chiederanno anche le righe da inserire nei file `language.dat`, `language.def` e `language.def.lua`.

5. La sigla ISO è una stringa di 2 o 3 caratteri che identifica il nome della lingua; vedi (WIKIPEDIA, 2003).

Purtroppo non posso indicare nessuna pubblicazione che permetta di conoscere le regole di sillabazione grammaticali; sembrerebbe che i linguisti che si occupano delle lingue più diverse non abbiano interesse per questo aspetto sicuramente secondario, ma certamente non irrilevante.

Avendo capito come funzionano questi vari file di descrizione, di pattern, di caricamento dei pattern nei file di formato, non sarà difficile, nello spirito di un software aperto com'è il sistema `TeX`, contribuire con il giusto feedback con gli autori dei pacchetti esistenti.

Riferimenti bibliografici

- ADRAENS, H. (2014). «The `xkeyval` package». <http://mirrors.ctan.org/macros/latex/contrib/xkeyval/xkeyval.pdf>.
- BERRY, K. (2015). «The `TeX` Live guide 2015». <https://www.tug.org/texlive/doc/texlive-en/texlive-en.pdf>.
- BEZOS, J. (2013). «CONTRIB». <http://ctan.mirror.garr.it/mirrors/CTAN/macros/latex/required/babel/base/CONTRIB>.
- (2015). «Babel». <http://mirrors.ctan.org/macros/latex/required/babel/base/babel.pdf>.
- CARETTE, F. e REUTENAUER, A. (2015). «Polyglossia: an alternative to Babel for `XLaTeX` and `LuaLaTeX`». <http://mirrors.ctan.org/macros/latex/contrib/polyglossia/polyglossia.pdf>.
- CTAN TEAM (2015.). «Upload to the Comprehensive TeX Archive Network (CTAN)». <https://www.ctan.org/upload?lang=en>.
- KNUTH, D. E. (1984). *The `TeX` book*. Addison Wesley Publ. Co., Reading, MA.
- ROBERTSON, W. e HOSNY, K. (2013). «The `XLaTeX` reference guide». <http://mirrors.ctan.org/info/xetexref/xetex-reference.pdf>.
- WIKIPEDIA (2003). «ISO 639». https://it.wikipedia.org/wiki/ISO_639.

▷ Claudio Beccari
claudio dot beccari at gmail
dot com

Questa rivista è stata stampata
presso Centro Stampa e Riproduzione S.r.l.,
Via di Pietralata, 157 – 00158 Roma,
su carta vellum white 80 g.
Copertina: vellum white 260 g.



Tredicesimo convegno nazionale su T_EX, L^AT_EX e tipografia digitale

Brescia, 29 ottobre 2016

Call for Papers

Il tredicesimo Convegno annuale su T_EX, L^AT_EX e tipografia digitale, organizzato dal Gruppo Utilizzatori Italiani di T_EX, si terrà a Brescia, il 29 ottobre, presso l'Università Cattolica, in via Trieste 17. Il Convegno sarà un momento di ritrovo e di confronto per la comunità L^AT_EX italiana, tramite una serie di interventi atti sia a contribuire all'arricchimento sia a supportarne lo sviluppo.

Maggiori informazioni sulla data del Convegno e sulle modalità di presentazione degli interventi saranno disponibili all'indirizzo:

<http://www.guitex.org/guitmeeting/2016/>

ArsT_EXnica

Rivista italiana di T_EX e L^AT_EX

Numero 21, Aprile 2016

- 3 Editoriale
Claudio Beccari
- 4 Generazione di file PDF conformi alle norme PDF/X e PDF/A con pdfT_EX—pdfx
C.V. Radhakrishnan, Hàn ThêThành, Ross Moore, Peter Selinger
- 13 La chimica con L^AT_EX 2_ε Un esame dei pacchetti esistenti e delle possibilità
Clemens Niederberger
- 21 Latin Challenges
Luigi Scarso
- 31 Primi esperimenti con `node` di LuaT_EX
Roberto Giacomelli
- 45 La gestione delle lingue e delle loro varianti
Claudio Beccari

