La gestione delle lingue e delle loro varianti

Claudio Beccari

Sommario

Le varianti linguistiche presentano alcuni problemi che l'utente normale del sistema 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 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 TEX system manages several languages and in part describes some methods to solve the above mentioned problems.

1 Introduzione

Usando LATEX con il programma *pdflatex* o con i programmi *xelatex* e *lulatex* 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 un 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 *pdflatex* si possono usare solo font codificati con non più di 8 bit; i font storici del sistema 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 *xelatex* che può gestire i font OpenType le cose sono un poco più semplici. Anche *lualatex* 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 *xelatex*, 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 UFT-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 `\`⟨*lingua*⟩`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 ⟨*lingua*⟩, viene automaticamente selezionata la famiglia di font `\`⟨*lingua*⟩`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 ⟨*lingua*⟩, babel e polyglossia hanno bisogno essenzialmente di un file chiamato rispettivamente ⟨*lingua*⟩`.ldf` e `gloss-`⟨*lingua*⟩`.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 ⟨*lingua*⟩. Se non esistesse un insieme di pattern associati alla ⟨*lingua*⟩, 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 ⟨*lingua*⟩ 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 ⟨*lingua*⟩ 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`⟨*lingua*⟩ 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 LaTeX, 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 lavorìo 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 TEX, 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 TEX sono attrezzati per questo scopo e svolgono questo lavoro con grandissima rapidità.

Va da se 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 don Donald Knuth nelle fasi iniziali della creazione del sistema TEX, produsse un programma *patgen* per creare i pattern a partire da una lista di 25 00 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 TEX; 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@ttribute{latin}{classic}{%
  \expandafter\addto\expandafter
  \extraslatin\expandafter{\extrasclassic}%
  }
\ifx\l@classiclatin\undefined
  \let\l@classiclatin\l@latin
  \PackageWarningNoLine{babel}{%
  Attention: hyphenation patterns
      for language\MessageBreak
  classiclatin 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@classiclatin}
```

L'attributo viene definito aggiungendo le impostazioni di `\extrasclassiclatin` a quelle di `\extraslatin`, ma poi verifica che la lingua `\l@classiclatin` 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@classiclatin` 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 equivalete a quello della lingua `\l@classiclatin` e questa equivalenza diventa permanente perché viene anche aggiunta alla lista delle impostazioni contenuta in `\extrasclassiclatin`.

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 dieresi;
- il greco moderno politonico che usa tutti i diacritici: gli spiriti e la dieresi; 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 divisone 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 ⟨*modificatori*⟩ con babel e con i ⟨*valori*⟩ nelle opzioni della forma `variant =` ⟨*valore*⟩ con polyglossia.

Per l'utente non c'è altro da fare se non ricordare che l'impostazione dei ⟨*modificatori*⟩ è permanente per l'intero lavoro di composizione; invece i ⟨*valori*⟩ 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 ⟨*chiave*⟩ `=` ⟨*valore*⟩, 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 chiamanti *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 *lualatex*; 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:

```
\initiate@active@char{:}%
\initiate@active@char{;}%
\initiate@active@char{!}%
\initiate@active@char{?}%
\declare@shorthand{french}{;}{%
    \ifhmode
    \ifdim\lastskip>\z@
        \unskip\penalty\@M\FBthinspace
        \else
          \FDP@thinspace
      \fi
    \fi
    \string;}
\declare@shorthand{french}{!}{%
    \ifhmode
      \ifdim\lastskip>\z@
        \unskip\penalty\@M\FBthinspace
      \else
        \FDP@thinspace
      \fi
    \fi
    \string!}
\declare@shorthand{french}{?}{%
    \ifhmode
      \ifdim\lastskip>\z@
        \unskip\penalty\@M\FBthinspace
      \else
        \FDP@thinspace
      \fi
    \fi
    \string?}
\declare@shorthand{french}{:}{%
    \ifhmode
      \ifdim\lastskip>\z@
        \unskip\penalty\@M\FBcolonspace
      \else
        \FDP@colonspace
      \fi
    \fi
    \string:}
\declare@shorthand{system}{:}{\string:}
\declare@shorthand{system}{!}{\string!}
\declare@shorthand{system}{?}{\string?}
\declare@shorthand{system}{;}{\string;}
\FB@addto{extras}{%
  \languageshorthands{french}%
  \bbl@activate{:}\bbl@activate{;}%
  \bbl@activate{!}\bbl@activate{?}%
}
\FB@addto{noextras}{%
  \bbl@deactivate{:}\bbl@deactivate{;}%
  \bbl@deactivate{!}\bbl@deactivate{?}%
}
```

```
\ifluatex
  \newluatexattribute\xpg@frpt %
  \directlua{polyglossia.load_frpt()}%
\else
  \newXeTeXintercharclass\french@punctthin % ! ? ; et autres
  \newXeTeXintercharclass\french@punctthick % :
  \newXeTeXintercharclass\french@punctguillstart % « ‹
  \newXeTeXintercharclass\french@punctguillend % » ›
\fi

\def\xpg@unskip{\ifhmode\ifdim\lastskip>\z@\unskip\fi\fi}
\def\xpg@nospace#1{#1}

\def\french@punctuation{%
    \lccode"2019="2019
    \ifluatex
      \global\xpg@frpt=1\relax %
      \directlua{polyglossia.activate_frpt()}%
    \else
      \XeTeXinterchartokenstate=1
      \XeTeXcharclass `\! \french@punctthin
      \XeTeXcharclass `\? \french@punctthin
      \XeTeXcharclass `\; \french@punctthin
      \XeTeXcharclass `\: \french@punctthick
      % ...
      \XeTeXcharclass `\« \french@punctguillstart
      \XeTeXcharclass `\» \french@punctguillend
      \XeTeXcharclass `\‹ \french@punctguillstart
      \XeTeXcharclass `\› \french@punctguillend
      \XeTeXinterchartoks\z@ \french@punctthin={\nobreak\thinspace}%
      \XeTeXinterchartoks\z@\french@punctthick={\nobreakspace}%
      \XeTeXinterchartoks255\french@punctthin={\xpg@unskip\nobreak\thinspace}%
      \XeTeXinterchartoks255\french@punctthick={\xpg@unskip\nobreakspace}%
      \XeTeXinterchartoks\french@punctguillstart\z@={\nobreakspace}% "«a" -> "« a"
      \XeTeXinterchartoks\z@\french@punctguillend={\nobreakspace}% "a»" -> "a »"
      \XeTeXinterchartoks\french@punctguillstart255={\nobreakspace\xpg@nospace}% "«  " -> "«~"
      \XeTeXinterchartoks255\french@punctguillend={\xpg@unskip\nobreakspace}% "  »" -> "~»"
      \XeTeXinterchartoks\french@punctguillend\french@punctthin={\nobreak\thinspace}% "»;" -> "» ;"
      \XeTeXinterchartoks\french@punctguillend\french@punctthick={\nobreakspace}% "»:" -> "» :"
      \XeTeXinterchartoks\french@punctthin\french@punctguillend ={\nobreakspace}% "?»" -> "? »"
    \fi
}
```

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' ottenuto 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 partici 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 TEX 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 TEX 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 parole[4] 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 analfabetici 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 XELATEX 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 TEXbook*. Addison Wesley Publ. Co., Reading, MA.

Robertson, W. e Hosny, K. (2013). «The XETEX 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`