HyPlain, più lingue insieme anche in Plain

Enrico Gregorio

Sommario

Si descrive un sistema per abilitare la cesura a fine riga in più lingue usando Plain TEX, insieme a un'interfaccia per definire lingue e le loro convenzioni.

Abstract

We describe a system for enabling hyphenation in several languages under Plain TEX, along with an interface to define the used languages and their conventions.

1 Introduzione

Nella giovinezza di TEX, cioè prima della versione 3, ogni formato poteva contenere l'insieme delle regole di cesura a fine riga per una sola lingua. Con un trucco piuttosto oscuro era in realtà possibile inserire le regole per due lingue, ma la faccenda era davvero complicata.

A quell'epoca LATEX era poco usato per lavori di matematica, dato che gli ambienti per il trattamento delle formule si riducevano a equation e eqnarray. Di buono aveva la numerazione automatica e i riferimenti incrociati; troppo poco per farlo preferire a $\mathcal{AMS}$-TEX, il formato basato su Plain creato da Michael Spivak per l'American Mathematical Society.

Come si faceva per avere le cesure corrette in italiano, visto che la versione standard di Plain forniva le regole per l'inglese americano? C'erano due modi: la prima soluzione era di ignorare il problema, con cesure inaccettabili come 'fas-tidio'. La seconda era di modificare il formato Plain in modo che caricasse le regole per la cesura dell'italiano che Claudio Beccari aveva prodotto.

Il formato Plain viene creato con iniTEX dal documento `plain.tex` il quale a un certo punto contiene la riga

```
\input hyphen
```

e bastava sostituirla con

```
\input ithyph
```

rinominando il documento come `plainit.tex`.

Ai tempi la riga di comando non faceva molta paura: non c'erano altri modi per lavorare, su certe piattaforme! Per chi usava il Macintosh c'erano OzTEX o *Textures* con i quali creare un nuovo formato era molto semplice.

È bene ricordare una delle particolarità di TEX. All'avvio del programma, con un metodo che dipende dal sistema operativo e dalla distribuzione, l'interprete TEX (che può anche essere PDFTEX) carica un *formato*, cioè uno stato iniziale. La maggior parte degli utenti usa LATEX, ma non si è certo ristretti a questo. La creazione del formato coinvolge l'uso di iniTEX, una versione speciale del programma, ed è eseguita automaticamente, di solito in fase di installazione; tuttavia a volte è necessario rifarla, per esempio proprio per abilitare nuove lingue con babel in LATEX. In questo articolo si farà riferimento a TEX, ma tutto vale anche per PDFTEX o XETEX.

Nel 1989 arrivò TEX versione 3 (KNUTH, 1989), con la possibilità di inserire in un singolo formato le regole di cesura anche per 256 lingue diverse e subito si cercò di sfruttare l'opportunità per creare formati compatibili con più lingue. Il documento più semplice su cui far girare iniTEX per ottenere un formato in cui sia possibile scrivere correttamente in inglese e in italiano è

```
\input plain
\language1
\input ithyph
\language0
\def\english{\language0 }
\def\italian{\language1 }
\dump
```

Con il formato creato da questo si possono avere le cesure corrette sia in inglese americano che in italiano, basta dichiarare all'inizio del proprio documento la lingua preferita. Si può anche avere un documento in cui le lingue coesistono, basta dichiarare la lingua in un gruppo.

Non è qui il luogo per un'analisi di come funziona la scelta delle regole di cesura; basti sapere che TEX tiene conto del valore assegnato alla variabile `\language` e che

```
\language=0
\language0
```

sono modi del tutto equivalenti di assegnare a questa variabile il valore zero. TEX associa un insieme di regole al valore della variabile; perciò, con l'esempio precedente, quando `\language` vale zero si useranno le regole inglesi (lette tramite `\input plain`), mentre quando vale uno si useranno quelle italiane. Conseguenza di questo è che per disabilitare le cesure è sufficiente assegnare il valore 255 a `\language`, visto che è alquanto improbabile che si siano definite le regole per 256 lingue,

questo vale anche in LaTeX. Tuttavia, se il valore è un intero non compreso fra 0 e 255, verranno usate le regole come per il valore 0. Si veda l'Appendice H del TeXbook.

È evidente che l'interfaccia proposta è troppo primitiva, in realtà occorre fare qualcos'altro; per esempio, per avere le cesure corrette con parole precedute dall'apostrofo in italiano è necessario dare il comando

```
\lccode`\'=`\'
```

perché TeX abbandona la sillabazione se in una sequenza di caratteri ne compare uno il cui `\lccode` sia zero.[1] Inoltre l'italiano permette di spezzare anche le ultime due lettere di una parola, a differenza dell'inglese, che ne richiede almeno tre sulla nuova riga. Dunque occorre che la definizione di `\italian` sia modificata con

```
\def\italian{\language1
  \lccode`\'=`\'
  \righthyphenmin2 }
```

Analogamente occorre ridefinire `\english` per disfare ciò che `\italian` può avere fatto:

```
\def\english{\language1
  \lccode`\'=0
  \righthyphenmin3 }
```

Ovviamente la situazione si complica di molto se invece di due sole lingue se ne hanno di più, ciascuna con le sue convenzioni e necessità. Si pensi al francese che richiede spaziature prima di certi segni di interpunzione e molte altre cose.

Daniel Flipo e Laurent Siebenmann crearono proprio per le esigenze del francese il pacchetto HyMaster (Siebenmann, 1993), che risolveva in modo decente certi problemi. A partire da quello mi creai un sistema meno complesso che poteva servire le mie esigenze e quelle dei colleghi, allora a Catania. Poi me ne dimenticai, perché nel frattempo LaTeX aveva incorporato, tramite il pacchetto amslatex, le funzionalità di $\mathcal{AMS}$-TeX. Inoltre babel (Braams, 1991) permetteva non solo di definire le regole di cesura per più lingue, ma anche di modificare 'al volo' le parole fisse.

Non c'era quindi più bisogno di rimanere ancorati a $\mathcal{AMS}$-TeX e la maggior parte dei matematici si trasferì a LaTeX, soprattutto con l'uscita di LaTeX 2_ε, nel 1994.

Fu con grande stupore che lessi sul forum del GuIT un intervento[2] in cui si chiedeva aiuto per installare un formato basato su Plain che riconobbi come le mie antiche macro! Da quella discussione si sviluppò l'idea di migliorare quel pacchetto, in modo da rendere più facile il passaggio da una lingua all'altra e offrire un'interfaccia per aggiungere codice da eseguire quando si passa a una lingua diversa.

Lo scopo di questo articolo non è tanto di descrivere un pacchetto di uso probabilmente molto limitato, quanto di scoprire qualcuno dei trucchi che ho usato e che possono essere utili anche in altre situazioni.

2 Il pacchetto HyPlain

Il pacchetto consiste di tre documenti: `hyplain`, `hyrules` e `hylang`, tutti con estensione `.tex`. Il primo è quello su cui si fa girare iniTeX, il secondo quello in cui si definisce l'interfaccia utente. Il terzo è modificabile dall'utente, che può scegliere quali lingue abilitare e definire le convenzioni da usare nelle varie lingue.

Non discuterò qui l'installazione del formato prodotto nelle varie distribuzioni, visto che chi usa Plain o i suoi derivati è abituato a mettere le mani a basso livello. Su un sistema Unix, per esempio con TeX Live, è sufficiente la seguente sequenza di comandi:

```
$ tex -ini hyplain
...
Beginning to dump on file hyplain.fmt
...
$ ln -s `which tex` hyplain
```

per avere nella directory sia il formato che un eseguibile richiamabile dalla linea di comando con `./hyplain`. Si può naturalmente trasferire il file `hyplain.fmt` e l'eseguibile in luoghi opportuni, quali siano dipende dalla distribuzione.

Su questo formato è possibile poi sovrapporre $\mathcal{AMS}$-TeX al modo usuale:

```
$ tex -ini -jobname hyamstex \&hyplain
...
* \input amstex
...
* \dump
...
Beginning to dump on file hyamstex.fmt
...
$
```

(con `$` indico l'invito della shell, con `*` quello di TeX e con `...` riassumo i messaggi inviati sullo schermo).

Il contenuto di `hyplain.tex` è molto semplice:

```
1 \catcode`\{=1
2 \catcode`\}=2
3 \catcode`\@=11
4
5 \let\@@input\input
6
7 \def\input hyphen {%
8   \let\input\@@input
```

1. La notazione `'\'` permette di riferirsi al codice ASCII del carattere che compare dopo la barra rovescia.

2. http://www.guit.sssup.it/phpBB2/viewtopic.php?t=1451

```
9   \input hyrules }
10
11 \@@input plain
12
13 \def\fmtname{plain %
14   (multiple language support)}
15 \def\hyplainversion{1.0}
16
17 \dump
```

Le righe 1 e 2 servono per definire le graffe come caratteri di inizio e fine dei gruppi, perché questo non è stabilito da iniTEX. La riga 3 serve per dichiarare `@` come lettera, al modo solito, per definire comandi interni.

Nella riga 5 memorizzo il significato di `\input`. Infatti voglio evitare che sia caricato il file `hyphen.tex`. Una delle consegne di Knuth è che il file `plain.tex` non sia modificato che da lui stesso; in questo file perciò ci sarà sempre (a meno che Knuth non cambi idea) la riga

```
\input hyphen
```

Con le righe 7–9 ridefinisco `\input` come una macro delimitata dalla stringa `hyphen␣`, il cui effetto è di ridare a `\input` il suo significato originale e di caricare `hyrules`.

A seguire carichiamo `plain.tex` e durante l'elaborazione di questo verrà eseguita la macro (ridefinita) `\input`, con il risultato detto prima.

Il resto del file è chiaro: si definiscono due macro che contengono un'identificazione del pacchetto e poi si esegue `\dump` che scrive il file `hyplain.fmt`.

3 hyrules.tex

Esaminiamo ora il secondo file del pacchetto, quello dove vengono introdotti i comandi dell'interfaccia per usare più lingue.

```
1 \chardef\myh@us@catcode=\the\catcode`\_
2 \catcode`\_=11
3 \let\@xp\expandafter
4 \newlinechar`^^J
```

Fin qui nulla di complicato: si salva il codice di categoria del carattere `_` che poi diventa una lettera, che userò per alcuni comandi interni. Si veda Gregorio (2006) per informazioni sui codici di categoria. Introduco anche un'abbreviazione per rendere meno pesante il codice che segue e poi definisco il carattere che, nei messaggi sullo schermo, provoca un 'a capo'.

Ogni lingua che sarà caricata con `hylang.tex` è identificata da una macro della forma `\`⟨*stringa1*⟩`_`⟨*stringa2*⟩; queste stringhe possono essere arbitrarie, ma consiglio di usare i codici ISO di lingua e nazione; per esempio, l'inglese americano sarà identificato da `\en_US`, l'italiano in Italia sarà `\it_IT` e l'italiano in Svizzera `\it_CH`.

Naturalmente l'utente non potrà scrivere direttamente questi comandi, a cui si accede indirettamente; infatti il codice di categoria di `_` ritornerà quello usuale.

Uno di questi comandi è riservato: `\zz_ZZ` indica una lingua *universale*, nella quale non sono definite regole di cesura e che verrà usata in mancanza di una definizione in `hylang`. Trovo che questa sia una strategia migliore di quella di babel, che invece usa le cesure dell'inglese americano per quelle lingue di cui TEX non possieda le regole.

Compariranno spesso costrutti del tipo

```
\@xp\let\@xp\foo\csname bar\endcsname

\@xp\edef\csname bar\endcsname{xyz}
```

in cui `bar` contiene i parametri `#1` e `#2`. Non c'è da prenderne paura: il primo equivale a

```
\let\foo\bar
```

perché il primo provoca l'espansione del secondo che a sua volta espande `\csname`. Il secondo equivale a

```
\edef\bar{xyz}
```

per motivi analoghi. Ricordo che `\edef` espande ciò che è contenuto nelle graffe, ma nel caso di `\the` seguito da un registro token si limita ad estrarne il contenuto. Inoltre

```
\csname ...\endcsname
```

produce una sequenza di controllo il cui nome consiste dei caratteri che risultano dall'espansione dei token compresi fra i due comandi, senza distinzione sui codici di categoria. È importante ricordare che dall'espansione devono risultare solo caratteri. Perciò, se l'argomento `#1` è `ab` e `#2` è `CD`, l'espansione di

```
\csname #1_#2\endcsname
```

produrrà un token equivalente alla sequenza (in genere illegale) `\ab_CD`.

```
5 \def\selectlanguage#1#2{%
6   \@xp\ifx\csname #1_#2\endcsname\relax
7     \global\@xp\let
8       \csname #1_#2\endcsname\zz_ZZ
9     \myh@nel{#1}{#2}%
10  \fi
11  \csname #1_#2\endcsname
12  \ignorespaces}
```

La macro `\selectlanguage`, come in babel, è quella che sceglie una nuova lingua. A differenza del comando di babel ha due argomenti, le due stringhe di cui si compone l'identificativo. Se si scrive

```
\selectlanguage{ab}{CD}
```

la macro esamina per prima cosa se il comando `\ab_CD` è definito. Le regole di TEX dicono che se

```
\csname ab_CD\endcsname
```

non corrisponde a un comando definito, il suo valore equivale a `\relax`; dunque, se `\ab_CD` non è definito, il condizionale è vero e si rende `\ab_CD` equivalente a `\zz_ZZ`, emettendo un messaggio di avviso. Poi il comando `\ab_CD` viene eseguito e ogni spazio che segua il secondo argomento di `\selectlanguage` è ignorato.

```
\def\myh@nel#1#2{%
  \message{^^Jhyplain warning:
    Language #1 for #2 undefined,
    using fallback instead^^J}}
```

La macro `\mhy@nel` contiene il messaggio di avviso che la lingua scelta non è definita, starà all'utente correggere l'errore.

```
\def\addalias#1#2#3{%
  \@xp\let\@xp#1\csname #2_#3\endcsname}
```

Questa macro è l'interfaccia per definire comandi abbreviati o più espliciti per la scelta di una lingua; per esempio si potrebbe dare, nel proprio documento o in `hylang`,

```
\addalias\IT{it}{IT}
\addalias\usenglish{en}{US}
```

Così `\IT` e `\usenglish` diventano equivalenti, rispettivamente, a

```
\selectlanguage{it}{IT}
\selectlanguage{en}{US}
```

Ora viene la parte più complessa del pacchetto: quando si sceglie una lingua si deve anche eseguire un certo codice, come abbiamo visto prima, e questo codice va anche 'neutralizzato' quando si passa a una lingua diversa.

Un modo potrebbe essere quello di scrivere il comando `\selectlanguage` o le abbreviazioni sempre dentro un gruppo; ma non si può contare su questo, che potrebbe anche diventare pesante per l'utente. Si pensi a un testo scritto prima in inglese e poi in italiano, per un lungo tratto, e poi ancora in inglese: occorrerebbe mettere un inizio di gruppo quando si comincia a scrivere in italiano e poi ricordarsi di chiuderlo e questo potrebbe portare a errori a volte incomprensibili.

Molto meglio definire un insieme di comandi da eseguire cominciando a scrivere in una lingua e il codice che annulla queste modifiche, da eseguire quando si comincia a scrivere in una lingua diversa. Se poi `\selectlanguage` è dato in un gruppo, le modifiche saranno annullate automaticamente. Li chiameremo, rispettivamente, codice *IN* e *OUT*; saranno ovviamente diversi per ciascuna delle lingue abilitate.

Una strategia analoga, e più complessa, è usata da `babel`, che però ha molte cose di cui tenere conto, come i testi fissi, i registri, i caratteri attivi e così via (Gregorio, 2007). Molti di questi aspetti vengono lasciati da HyPlain all'utente, che si suppone più addentro alla programmazione TEX di quanto non sia quello normale di LATEX.

In tutto quanto segue supporremo sempre che la lingua da considerare abbia l'identificativo `\ab_CD`. Oltre a questa macro ne vengono definite altre due, `\extras@ab_CD` e `\noextras@ab_CD`; la prima contiene il codice *IN* di quella lingua, la seconda il codice *OUT*.

```
\newtoks\myh@toks
\def\addto#1#2#3#4{%
  \@xp\let\@xp\myh@temp
    \csname extras@#1_#2\endcsname
  \myh@toks=\@xp{\myh@temp}%
  \myh@toks=\@xp{\the\myh@toks #3}
  \@xp\edef\csname extras@#1_#2\endcsname
    {\the\myh@toks}%
  \@xp\let\@xp\myh@temp
    \csname noextras@#1_#2\endcsname
  \myh@toks=\@xp{\myh@temp}%
  \myh@toks=\@xp{\the\myh@toks #4}
  \@xp\edef\csname noextras@#1_#2\endcsname
    {\the\myh@toks}%
}
```

Si comincia allocando un registro token. La macro `\addto` ha quattro argomenti:

1. la prima parte dell'identificativo,
2. la seconda parte dell'identificativo,
3. il codice 'in ingresso', che denoteremo con *IN*,
4. il codice 'in uscita', che denoteremo con *OUT*.

Si definisce una macro temporanea equivalente a `\extras@ab_CD`, poi si carica il registro token con la sua espansione di primo livello; l'istruzione successiva aggiunge al contenuto del registro il terzo argomento, cioè il codice *IN*, e, finalmente, si ridefinisce `\extras@ab_CD` con il contenuto di questo registro.

Per non essere troppo astratti, l'effetto sarà questo, se `\extras@ab_CD` vale `\foo\bar=2`␣ e il codice *IN* è `\xyz=9`␣: alla fine del processo, sarà come avere dato

```
\def\extras@ab_CD{\foo\bar=2 \xyz=9 }
```

Lo stesso si fa al comando `\noextras@ab_CD` con il codice *OUT*. Il comando ha quindi effetto cumulativo.

```
\def\definebaselanguage#1#2#3{%
  \message{***^^JDefining language
    #1 for #2 using^^J}%
  \@xp\chardef\csname l@#1_#2\endcsname=0
  \language\csname l@#1_#2\endcsname
```

```
39   \input #3%
40   \message{This is \string\language
41     \@xp\the\csname l@#1_#2\endcsname
42     ^^J***^^J}%
43   \myh@initialize{#1}{#2}}
44 \def\definelanguage#1#2#3{%
45   \message{***^^JDefining language
46     #1 for #2 using^^J}%
47   \csname newlanguage\@xp\endcsname
48     \csname l@#1_#2\endcsname
49   \language\csname l@#1_#2\endcsname
50   \input #3%
51   \message{This is \string\language
52     \@xp\the\csname l@#1_#2\endcsname
53     ^^J***^^J}%
54   \myh@initialize{#1}{#2}}
```

I due comandi definiti in questa parte di codice sono molto simili. In effetti il primo si usa una volta sola per definire la lingua di base che è opportuno sia l'inglese americano.

A ogni lingua è associato un intero fra 0 e 255, il cui valore è contenuto nel comando `\l@ab_CD`; la differenza fra il primo e il secondo comando è che il primo associa alla lingua che si definisce l'intero 0, mentre il secondo impiega il meccanismo di allocazione definito in Plain TEX tramite `\newlanguage`. Ci sono tre argomenti:

1. la prima parte dell'identificativo,
2. la seconda parte dell'identificativo,
3. il nome del file che contiene le regole per la cesura.

Si comincia associando alla lingua l'intero; se la macro è chiamata tramite

```
\definelanguage{ab}{CD}{abhyphen}
```

l'effetto complessivo (oltre all'emissione di un paio di messaggi) è di eseguire

```
\newlanguage\l@ab_CD
\language\l@ab_CD
\input abhyphen
\myh@initialize{ab}{CD}
```

Nel caso di `\definebaselanguage` il primo comando eseguito sarebbe

```
\chardef\l@ab_CD=0
```

Infatti un comando definito tramite `\chardef` si comporta come un numero quando TEX se ne aspetta uno.

È importante emettere `\language\l@ab_CD` prima di caricare il file delle regole di cesura, perché iniTEX memorizza queste regole nel formato proprio associandole al valore della variabile `\language` in vigore. Discuteremo fra poco l'ultimo comando, che serve a eseguire gli opportuni codici *OUT* e *IN*.

C'è una curiosità alle righe 15–16; il comando `\newlanguage` è definito in Plain come `\outer`, quindi non può comparire così com'è nel testo di sostituzione in `\def`. Con il trucco di immergerlo in `\csname...\endcsname`, TEX non *vede* il comando proibito. Inoltre possiamo sfruttare il fatto che ciò che compare fra `\csname` e `\endcsname` viene espanso; perciò l'`\expandafter` provoca l'espansione del comando che segue (`\csname`), risparmiandoci un po' di lavoro; l'effetto sarebbe equivalente, nel solito esempio, a

```
\newlanguage\l@ab_CD
```

```
55 \def\definedialect#1#2#3#4{%
56   \@xp\let\csname l@#1_#2\@xp\endcsname
57     \csname l@#3_#4\endcsname
58   \myh@initialize{#1}{#2}}
```

Alcune lingue possono condividere le regole di cesura; l'esempio classico è il tedesco parlato in Germania e in Austria: sono la stessa lingua, ma con convenzioni leggermente diverse. Per esempio, in Germania il mese di gennaio si chiama *Januar*, in Austria è *Jänner*. Con

```
\definedialect{ab}{CD}{ab}{EF}
```

si definisce effettivamente una nuova macro `\ab_EF` con le due associate a cui assegnare diversi codici *IN* e *OUT*, ma questo dialetto si comporta come la lingua di base per quanto riguarda le regole di cesura.

```
59 \def\refinelanguage#1#2#3#4{%
60   \@xp\def\csname extras@#1_#2\endcsname
61     {#3}%
62   \@xp\def\csname noextras@#1_#2\endcsname
63     {#4}%
64 }
65 \let\refinedialect=\refinelanguage
```

Nel file `hylang` l'utente dovrebbe provvedere i codici *IN* e *OUT* iniziali per le lingue che definisce. Questo viene fatto tramite `\refinelanguage`. L'utente potrà aggiungere codice tramite `\addto` nel proprio documento. Gli argomenti sono come già descritto per `\addto` e non c'è bisogno di altre spiegazioni.

```
66 \def\myh@initialize#1#2{%
67   \@xp\def\csname #1_#2\endcsname{%
68     \language\csname l@#1_#2\endcsname
69     \myh@undo
70     \def\myh@lan{#1_#2}%
71     \csname extras@#1_#2\endcsname}%
72   \@xp\def\csname extras@#1_#2\endcsname{}%
73   \@xp\def\csname noextras@#1_#2\endcsname{}}
74 \def\myh@undo{%
75   \csname noextras@\myh@lan\endcsname}
```

La macro `\myh@initialize` è la principale del pacchetto. I suoi argomenti sono le componenti dell'identificativo della lingua. Con `\myh@initialize{ab}{CD}` vengono eseguiti i seguenti comandi:

```
\def\ab_CD{%
  \language\l@ab_CD
  \myh@undo
  \def\myh@lan{ab_CD}%
  \extras@ab_CD}
\def\extras@ab_CD{}
\def\noextras@ab_CD{}
```

Si definiscono cioè le macro `\ab_CD` e le due associate. Quando poi `\selectlanguage{ab}{CD}` esegue la macro `\ab_CD`, succedono le seguenti cose:

1. si passa alla lingua prescelta;
2. si esegue la macro `\myh@undo`;
3. si definisce la macro `\myh@lan` che contiene l'identificativo della lingua prescelta;
4. si esegue il codice *IN* relativo.

Un'altra curiosità è la mancanza di `\expandafter` prima di `\language`; questo comando è primitivo ed è uno di quelli che si aspettano di essere seguiti da un numero, quindi provocano l'espansione del token successivo. Analogamente, alla riga 7, `\global` è una primitiva che deve essere seguita da un comando di assegnazione e quindi anch'essa espande il token successivo.

La macro `\myh@undo` è definita come

```
\csname noextras@\myh@lan\endcsname
```

e, quando è eseguita, causa l'espansione di `\myh@lan`, quindi produce l'identificativo della lingua precedentemente in uso, come abbiamo visto poco fa e quindi l'esecuzione del codice *OUT* della lingua che si sta abbandonando.

Facciamo un paio di esempi. Supponiamo che l'utente scriva

```
\selectlanguage{ab}{CD}
testo
\selectlanguage{ef}{GH}
altro testo
\selectlanguage{ab}{CD}
```

Il primo comando esegue `\ab_CD` e quindi il significato di `\myh@lan` è, a questo punto, `ab_CD`. Quando viene eseguita la scelta della seconda lingua, prima di `\extras@ef_GH` viene espansa `\myh@undo`, cioè precisamente la macro `\noextras@ab_CD`. In seguito il comando `\myh@undo` si espanderà, come si desidera, in `\noextras@ef_GH`, per annullare, al terzo `\selectlanguage`, gli effetti del codice *IN* relativo alla lingua `\ef_GH`.

Se invece l'utente scrive

```
\selectlanguage{ab}{CD}
testo
{\selectlanguage{ef}{GH}
altro testo}
```

il testo nel gruppo sarà composto secondo le regole valide per la lingua `\ef_GH`. La macro `\myh@undo` verrà ridefinita all'interno del gruppo, ma alla sua chiusura riotterrà il valore precedente.

Dunque, in ogni caso, gli effetti del codice *IN* relativo alla seconda lingua scelta, verranno annullati, per effetto di comandi espliciti oppure per via dell'uscita da un gruppo.

```
76 \def\hyphenmins#1#2{%
77   \lefthyphenmin=#1\relax
78   \righthyphenmin=#2\relax}
79
80 \input hylang
81
82 \let\definebaselanguage\undefined
83 \let\definelanguage\undefined
84
85 \def\myh@lan{} % initialization
86 \def\noextras@{} % initialization
87 \en_US
88
89 \catcode`\_=\myh@us@catcode
```

La fine del file `hyrules` contiene cose semplici; il primo comando è l'interfaccia per definire la minima lunghezza dei tronconi di parola dopo la cesura; andrà usato nel codice *IN* di ogni lingua e non sarà quindi necessario annullarne gli effetti nel codice *OUT*.

A questo punto viene caricato `hylang.tex` di cui ci occuperemo in seguito. Poi i comandi per le definizioni di nuove lingue sono disabilitati, perché non ha senso farlo se non quando si compila con iniTEX, e si definiscono due comandi per non far arrabbiare TEX all'esecuzione di `\en_US`. Infatti a questo punto non c'è alcuna lingua precedente e, se si guarda il codice di cui abbiamo discusso finora, ci si rende conto della necessità di queste definizioni.

Per ultimo, il codice di categoria di `_` viene riportato a quello in vigore quando si è cominciato.

4 hylang.tex

Questo è il documento modificabile dall'utente a seconda delle sue esigenze. Qui vengono caricate le lingue che si vogliono usare; se si desidera cambiare la scelta, occorrerà modificare questo file e ricreare il formato `hyplain.tex`.

```
1 %%% US English must always come first
2 \definebaselanguage{en}{US}{hyphen}
3 \definelanguage{zz}{ZZ}{zerohyph}
4 %%% don't modify the lines above
5 \refinelanguage{en}{US}{\hyphenmins{2}{3}}{}
6 \refinelanguage{zz}{ZZ}{}{}
7
8 %%% Italian
9 \definelanguage{it}{IT}{ithyph}
10 \refinelanguage{it}{IT}
11   {\hyphenmins{2}{2}\lccode`\'=`\'}
12   {\lccode`\'=0 }
```

```
%%% Add other languages if needed

%%% Aliases
\addalias\US{en}{US}
\addalias\IT{it}{IT}
\addalias\ZZ{zz}{ZZ}
\addalias\nohyphens{zz}{ZZ}
```

Le prime due righe non commentate sono quelle per definire la lingua di base che è, come già detto, l'inglese americano. Poi vediamo un esempio per l'italiano, con l'impostazione dei minimi di cesura e il cambiamento del codice `\lccode` per l'apostrofo.

Nelle moderne distribuzioni è presente il file `zerohyph.tex` che non definisce alcuna regola di cesura. Scegliendo questa 'lingua' si inibisce del tutto la sillabazione a fine riga. Come detto, questa è la lingua 'universale' che viene scelta se si dà un comando per una lingua sconosciuta.

Per finire, alcuni sinonimi: per esempio, invece di `\selectlanguage{it}{IT}` è possibile scrivere semplicemente `\IT`.

5 Commenti

L'interfaccia prevista è molto primitiva, ma ampliabile con una certa facilità. Supponiamo, per esempio, che si voglia, come nel modulo italiano di babel modificare i valori di `\clubpenalty` e `\widowpenalty`. Occorrerà modificare anche il codice *IN* per le altre lingue, in modo che i valori desiderati per esse siano ripristinati al cambio di lingua. Vogliamo anche che in italiano si usi la spaziatura 'francese', cioè che gli spazi dopo i segni di interpunzione siano come quelli fra le parole.[3]

Il contenuto di `hylang` dovrà diventare (togliendo le cose superflue)

```
\definebaselanguage{en}{US}{hyphen}
\definelanguage{zz}{ZZ}{zerohyph}

\refinelanguage{en}{US}
  {\hyphenmins{2}{3}%
   \clubpenalty150
   \widowpenalty150 }
  {}

\refinelanguage{zz}{ZZ}
  {\clubpenalty150
   \widowpenalty150 }
  {}

\definelanguage{it}{IT}{ithyph}
\refinelanguage{it}{IT}
  {\hyphenmins{2}{2}\lccode`\'=`\'
   \clubpenalty3000
   \widowpenalty3000
   \frenchspacing}
  {\lccode`\'=0
   \nonfrenchspacing}
```

3. Qualcuno obietta che non è questa la spaziatura francese; in effetti Knuth, con questo nome, intendeva riferirsi solo agli spazi *dopo* i segni di interpunzione.

L'utente che usa Plain è in genere abbastanza smaliziato per accorgersi di queste sottigliezze. Va in *OUT* solo il codice di cui si sa che annulla ciò che si produce con il codice *IN*. I valori dei parametri interni vanno sempre specificati nel codice *IN* e, se li si cambia in una lingua, occorre modificarli in tutte.

6 Sviluppi futuri

In babel c'è un meccanismo per conservare i valori delle variabili e dei comandi, probabilmente sarà incluso nella prossima versione di HyPlain. L'idea di babel è di mantenere in un comando il codice da eseguire a ogni cambio di lingua per ritornare a uno stato 'iniziale'.

Dovrebbe essere sufficiente aggiungere l'espansione di questo comando, che chiamerò `\myh@saved`, a ogni `\selectlanguage`, in modo da rimettere a posto valori delle variabili e significati delle macro modificate.

```
\def\myh@saved{}
\newcount\myh@savecnt
\def\hyplainsave#1{%
  \@xp\let\csname myh@%
    \number\myh@savecnt\endcsname #1\relax
  \begingroup
    \toks@\@xp{\myh@saved \let#1=}%
    \edef\x{\endgroup
      \def\noexpand\myh@saved{%
        \the\toks@ \@xp\noexpand
          \csname myh@\number\myh@savecnt
            \endcsname\relax}}%
  \x
  \advance\myh@savecnt\@ne}
\def\hyplainsavevariable#1{\begingroup
    \toks@\expandafter{\myh@saved #1=}%
    \edef\x{\endgroup
      \def\noexpand\myh@saved{%
        \the\toks@ \the#1\relax}}%
  \x}
```

È interessante vedere come funziona questo codice. Supponiamo di scrivere `\hyplainsave\pippo` e che il valore del contatore `\myh@savecnt` sia 25. La prima cosa che viene eseguita è

```
\let\myh@25\pippo
```

La sequenza `\myh@25` è illegale, se scritta così, ma in realtà accediamo a essa tramite `\csname...\endcsname` e, una volta che è entrata nel meccanismo di lettura e assorbita da TEX non è diversa dalle altre.

Nel seguito indicheremo con `<myh@saved>` il testo di sostituzione della macro `\myh@saved`.

Viene poi aperto un gruppo e in esso eseguito

```
\toks@\@xp{\myh@saved \let\pippo=}
```

cosicché il registro token temporaneo \toks@ contiene

```
{<myh@saved> \let\pippo=}
```

Ora viene il bello. Si esegue un \edef che espande tutto ciò che c'è tra le graffe. Quindi questo diventa equivalente a

```
\def\x{\endgroup
  \def\myh@saved{<myh@saved>
    \let\pippo=\myh@25\relax}
```

La successiva esecuzione di \x ridefinisce \myh@saved che quindi conterrà il codice necessario a ridare a \pippo il suo significato originale. A questo punto il contatore viene fatto scattare di uno, in modo che il prossimo uso di \hyplainsave usi un comando diverso dal precedente, in questo caso \myh@27. Il gruppo viene chiuso durante l'espansione di \x, così TEX dimentica la definizione di questo e i valori assegnati nel gruppo.

Con \hyplainsavevariable la faccenda è analoga; se \pluto è un registro dimensionale il cui valore è 42 pt, il registro \toks@ conterrà

```
{<myh@saved> \pluto=}
```

e la successiva definizione (con \edef) di \x equivale a

```
\def\x{\endgroup
  \def\myh@saved{<myh@saved>
    \let\pluto=42pt\relax}
```

In questo modo dovrebbe essere possibile far diventare il file hyrules dell'esempio precedente

```
\definebaselanguage{en}{US}{hyphen}
\definelanguage{zz}{ZZ}{zerohyph}

\refinelanguage{en}{US}
  {\hyphenmins{2}{3}}{}

\refinelanguage{zz}{ZZ}{}{}
\definelanguage{it}{IT}{ithyph}
\refinelanguage{it}{IT}
  {\hyphenmins{2}{2}\lccode`\'=`\'
   \clubpenalty3000
   \widowpenalty3000
   \frenchspacing}
  {\lccode`\'=0
   \nonfrenchspacing}

\hyplainsavevariable\clubpenalty
\hyplainsavevariable\widowpenalty
```

con evidente economia di pensiero: basta dichiarare tutte le variabili o comandi che si vogliono preservare. Questi comandi potranno anche essere dati nel documento dell'utente, cambiando i valori per mezzo di \addto.

Ogni suggerimento o richiesta di nuove funzioni è bene accetto: ogni tanto è bene giocare un po' con Plain TEX.

Riferimenti bibliografici

Braams, J. (1991). «Babel, a multilingual style-option system for use with LaTeX's standard document styles». *TUGboat*, **12** (2), pp. 291–301.

Gregorio, E. (2006). «Codici di categoria». *ArsTEXnica*, **1**, pp. 5–14.

— (2007). «Enjoying babel». *TUGboat*, **28** (2), pp. 247–255.

Knuth, D. E. (1989). «The new versions of TEX and METAFONT». *TUGboat*, **10** (3), pp. 325–328.

Siebenmann, L. (1993). «A format compilation framework for European languages». *TUGboat*, **14** (3), pp. 212–221.

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