Implementation Tricks in the Hungarian Babel Module

Péter Szabó
Department of Analysis
Budapest University of Technology and Economics
Műegyetem rakpart 3–9.
Budapest H-1111
Hungary
pts+tug@math.bme.hu

Abstract

magyar.ldf, the Hungarian Babel module, was rewritten in the autumn of 2003 to obey most of the Hungarian typographical rules. This article describes some implementation issues, TeX macro programming hacks, and LaTeX typesetting trickery used in magyar.ldf. All features of the new magyar.ldf are enumerated, but only those having an interesting implementation are presented in detail. Most of the tricks shown are useful for developing other language modules.

1 The Name of the Language

Usually a Babel language module has the English name of that language. For example, the German module is called germanb.ldf, and not deutsch.ldf. The Hungarian module is an exception to this rule, because it has the name magyar.ldf, in which "magyar" is the Hungarian adjective meaning "Hungarian". A similar exception is portuges.ldf for Portuguese. The letter "a" in word *magyar* has to be pronounced as in *blah*, and the consonant "gy" is the same as "d" in *due*.

The name of a language that a Babel language module (`.ldf` file) defines is usually specified as an argument of `\LdfInit` in the file. Thus, if czech.ldf is renamed to `foo.ldf`, it will have to be loaded with `\usepackage[foo]{babel}`, but to activate it, `\selectlanguage{czech}` should be used. This is not the case with magyar.ldf, because it detects its loaded filename using the `\CurrentOption` macro set by the `\ProcessOptions` command called from babel.sty. So whatever magyar.ldf is renamed to, that name is the one to pass to `\selectlanguage`.

The only reason why someone may want to rename an `.ldf` file is to load two different versions in the same LaTeX run. This is possible with magyar.ldf, but the user should be aware that the control sequences defined by the two copies will interact with each other in an unpredictable way. Experiments have shown that it is possible to load two copies of magyar.ldf with different load options (this is the so-called dual load):

```
\PassOptionsToPackage{frenchspacing=yes}
   {magyar.ldf}
\PassOptionsToPackage{frenchspacing=no}
   {hungarian.ldf}
\usepackage[hungarian,magyar]{babel}
```

Despite the name `hungarian.ldf` above, the file magyar.ldf gets loaded twice, because Babel translates the language name `hungarian` to file name magyar.ldf, and magyar.ldf expects options for `\CurrentOption.ldf`, which depends on the language name passed to `\usepackage[...]{babel}`. Since the dual load feature of magyar.ldf is experimental, most of the load options cannot be different in the two copies. So the safest way to load two copies is to replace the occurrences of the word `magyar` in the second copy with something else.

The latest magyar.ldf (version 1.5) is not part of standard Babel yet, but it is available as part of MagyarLaTeX (see section 4.1). Most of the typographical rules it tries to obey and problems it addresses were proposed in [1].

2 What an `.ldf` File Contains

An `.ldf` file is a Babel language module, which contains specific macros for the given language. It is loaded by babel.sty in the document preamble, at the time babel.sty itself is loaded. The macros defined in `foo.ldf` take effect only after changing the language with `\selectlanguage{foo}`. The default language is the one specified last in the `\usepackage[...] {babel}` command.

Babel itself contains the standard versions of the `.ldf` files as `tex/generic/babel/*.ldf`. In Babel 3.7 there are 41 of them; most are smaller than 10 kB. The largest files are: the old magyar.ldf defining the Hungarian language (25 kB), frenchb.ldf defining the French language (23 kB), spanish.ldf (21 kB), bulgarian.ldf (13 kB), ukraineb.ldf defining the Ukrainian language (12 kB), russianb.ldf (12 kB) and greek.ldf (9 kB). The new version of magyar.ldf is much larger than any of these: it is 178 kB. The size implies much more functionality, including several features unique to this new magyar.ldf — they will be discussed later in this document. Let's proceed first by dealing with features common in most `.ldf` files.

2.1 Selecting the Hyphenation Pattern Set

`foo.ldf` must define the control sequence `\l@foo` to be a number (`\newcount`, `\chardef`, etc.) representing the hyphenation pattern set to be used for that language. The language selection macro `\selectlanguage{foo}` calls `\language=\l@foo`, which activates the hyphenation patterns for the language `foo`.

The patterns were (presumably) defined with the `\patterns` primitive at the time iniTeX was called to generate the format file. The exact filename containing the `\patterns` command is specified in the file `language.dat`. If there is a line "`foot fthypen.tex`" in `language.dat`, then `\language =\l@foot` will activate `\patterns` found in `fthyph. tex`. In that case, `foo.ldf` should contain a line `\let\l@foo\l@foot`. But this line is omitted in most actual `.ldf` files, because the Babel language name and the hyphenation pattern set name is the same (`language.dat` would contain an entry starting with `foo␣` in our example). Note that the file `fthyph.tex` is read by iniTeX, not LaTeX, so the format files have to be re-generated each time `fthyph. tex` is changed.

Three different hyphenation pattern sets have been proposed for the Hungarian language (namely, `huhyph3.tex`, `huhyphc.tex` and `huhyphf.tex`). All of them are maintained by Gyula Mayer [4]. The most important difference among them is the way they hyphenate at subword boundaries of compound words. The document author can select any of these three by providing the appropriate load option to magyar.ldf (discussed later). The options work by redefining `\l@magyar` to be one of `\l@magyarf`, `\l@magyarc`, or `\csname l@magyar3\endcsname`.

There are two different correct ways to hyphenate compound words in Hungarian. magyarf hyphenates the most common foreign compound words of Hungarian text phonetically (e.g. *szink-ron*, meaning synchronous), while magyarc hyphenates them on the subword boundary (e.g. *szin-kron*). magyar3 is the old version of the hyphenation patterns which hyphenates most composite words phonetically (even non-foreign ones), save only a few exceptions listed explicitly. However, in all the three cases, hyphenation of foreign words cannot be perfect, because *all* of them cannot be specified in `\patterns`.

magyar.ldf redefines `\l@magyar` depending on the `hyphenation=` load option. If a given pattern set may be missing from the user's system, magyar.ldf falls back to another set with a meaningful warning message. Hyphenation is disabled not by choosing `\language0`, as Babel does, because `\language0` may contain valid patterns for a different language, but rather `\language255`, which is very likely to be unused since LaTeX assigns `\language` numbers from zero.

2.2 Defining Captions

LaTeX generates some words and phrases automatically. For example, `\tableofcontents` should emit the phrase "Table of contents" in the native language. The same applies for `\caption`s of figures and tables, and also for `\chapter` titles. Thus Babel expects `foo.ldf` to define a macro called `\captions foo` containing definitions like `\def\abstractname {Absztrakt}`. These definitions are executed by `\selectlanguage` each time the language is activated. So it is possible to have an English and then a Hungarian chapter in a book numbered 'Chapter 1' and '2. fejezet', respectively:

```
\chapter{foo} ...
\selectlanguage{magyar}\chapter{bar}
```

magyar.ldf has the proper definitions of Hungarian phrases. Some words contain accented letters, which are specified as commands (e.g. `\'a` for *á*) and not single 8-bit characters, so their interpretation does not depend on the active input encoding, i.e. the load option of inputenc.sty.

2.3 Generating Dates

`foo.ldf` should define a macro `\datefoo` to define the macro `\today`, which emits a date (specified by the `\year`, `\month`, `\day` registers) correctly for that language. The month name should be printed as a non-abbreviated word. The definition of `\today` is used by `\@date` invoked in `\maketitle` in the standard document classes.

In addition to defining `\today`, magyar.ldf defines the macro `\ondatemagyar` to further define `\ontoday`, which emits the date with the Hungarian equivalent of English *on*. The Hungarian language has suffixes instead of prepositions, and each suffix has several forms which must follow the vowel harmony of the word it is suffixed to. Thus "on March

15" is emitted as *március 15-én*, but "on March 16" is *március 16-án*, showing that the *-án/-én* suffix has two forms.

2.4 Minimum Hyphenation Length

TEX won't insert an implicit hyphen into the first `\lefthyphenmin` characters of words, nor in the last `\righthyphenmin` characters. The default TEX values for these are 2 and 3, respectively, which are suitable for the English language. `foo.ldf` can override the default by defining the macro `\foohyphenmins` to be *lr*, two digits specifying the left and the right minimum, respectively.

What magyar.ldf does depends on its load options. The default is to follow Hungarian typography: `\def\magyarhyphenmins{22}`.

Nine of the 41 `.ldf` files in Babel 3.7 do only the customizations described to this point. 25 languages go a little beyond these, and 7 languages go much beyond. Of those 25 + 7 languages that go beyond, we will compare frenchb.ldf in detail to magyar.ldf, because French and Hungarian share some typographical rules.

2.5 Defining Special Letters

Many languages have letters that are missing from the standard OT1 encoding, and some characters are missing even from T1. These should be implemented in `.ldf` files as control sequences. It is a common practice to modify the meaning of an existing letter, for example `czech.ldf` contains `\DeclareTextCompositeCommand{\v}{OT1}{t}{...}`. However, this declaration is contained in `\AtBeginDocument`, so they are in effect even when not the Czech language is active. This should have been avoided.

The correct solution is to use the *extras* facility provided by Babel: `foo.ldf` can have a macro `\extrasfoo`, which is executed each time the language foo is activated; and the macro `\noextrasfoo` is executed when the active language is about to change (because of a `\selectlanguage` command or when the end-of-group is reached). It is a common practice in `\extrasfoo` to save the meaning of a macro with `\babel@save`, or a meaning of a count, dimen or skip register with `\babel@savevariable`. The saved meanings will be restored just after `\noextrasfoo` is executed. Babel provides the command `\addto` that can append tokens to the definition of an existing macro. The idiom `\addto\extrasfoo{\babel@save\bar \def\bar{foo-bar}}` is typical, which gives a new meaning to `\bar` while the language foo is active.

The macro to be saved for `\DeclareTextCompositeCommand{\v}{OT1}` is `\OT1\v` (with the second backslash being part of the control sequence), but assigning the new meaning would be problematic, since `DeclareTextCompositeCommand` can be used only in the preamble. Thus the correct solution would involve fiddling with undocumented LATEX internals; which is probably why czech.ldf contains the problematic workaround using `\AtBeginDocument`.

Fortunately, the only non-English letters in the Hungarian language are accented vowels (*á*, *é*, *í*, *ó*, *ö*, *ő*, *ú*, *ü* and *ű*), which are all part of the T1 encoding. The letters *ő* and *ű* with the special Hungarian double-acute accent are missing from the Latin-1 encoding (ISO-8859-1), but are part of Latin-2. So authors dealing with Hungarian are encouraged to use `\usepackage[latin2]{inputenc}`.[1] `\usepackage{t1enc}` is also recommended, so TEX will be able to hyphenate words containing accented letters.

The finest Hungarian books have accents lowered a little bit. This is accomplished for the dieresis accent (¨) by calling the `\umlautlow` command (defined by Babel) in `\extrasmagyar`. No serious attempt is made to make this work for all three Hungarian accents, because the technology `\umlautlow` is based on works only for the OT1 encoding (which composes accented letters), but most Hungarian texts use the T1 encoding to allow hyphenation in words with accented letters.

The lowering of accents is possible using virtual fonts. But TEX font families come with too many variations and design sizes, so the virtual font generation would need to be automated. The macro `\lower@umlaut` in babel.def lowers accents by forcing their top to be 1.45 ex above the baseline. The `\accent` primitive lowers its accent by `\fontdimen5\font-1ex`, so the top of the accent can be forced to 1.45 ex by setting `\fontdimen5\font:=\ht0 - 0.45ex`, where `\ht0` is the height of the accent character (`\char127` in the OT1 encoding).

The lowering, in the case of *ü*, is as small as 0.43558 pt. Even this tiny displacement can make a visible difference: "ü < ü". The lowering method could be made adaptive by rendering the glyphs involved at high resolution, measuring the number of pixels between the accent and the letter vertically, and then lowering the accent so the distance will be a prescribed constant value.

[1] The most common incorrect letters found in Hungarian texts are *õ* and *û*: their presence is caused by software incapable of using Unicode or the Latin-2 encoding. These letters can be seen even on some huge advertisement banners on streets in Hungary. These texts were not typeset by TEX, of course!

Neither home users nor professionals use lowered accents in Hungary today, not even with books created with LaTeX — the original fonts with the T1 encoding are acceptable enough not to bother changing. TypoTeX Ltd., one of the biggest Hungarian publishing houses using TeX, developed the OM fonts in the early 1990s for use with plain TeX. The OM fonts are a variation of CM fonts with Hungarian accented glyphs added (with lowered accents). However, it is not worth creating `.fd` files for the OM fonts for use with LaTeX, because with the same amount of work new virtual fonts could be created from the EC fonts, which would take advantage of the full T1 character set, and existing, hinted fonts in Type 1 formats (such as the CM-Super fonts).

2.6 Hyphenation of Long Double Consonants

Hyphenating long double consonants in Hungarian is a difficult typographical problem. For example, the correct way to hyphenate the double consonant *tty* = *ty* + *ty* in the word *hattyú* ("swan") is *haty-tyú*. (There is a similar problem in German with words containing *ck*; [5] documents more languages with more exceptions.) The long double consonants involved are: *ccs*, *ddz*, *ddzs*, *ggy*, *nny*, *ssz*, *tty* and *zzs*. TeX's automatic hyphenation algorithm cannot deal with such exceptions, but adding ligatures dealing with the dash inserted by the implicit hyphenation can solve the problem. The simple trick of having `\patterns{t1ty}` and `t+-` → `ty-`[2] seems to solve the problem, because it hyphenates *tty* as *ty-ty*, but it also inserts an extra *y* before the hyphen in *fut-szalad*. Normal patterns will also insert an implicit hyphen into *botcsinálta*, yielding *bot-csinálta*. The ligature program above would then incorrectly alter that to *boty-csinálta*.

So a more elaborate set of ligatures would have to be constructed, to detect the context of the hyphen and insert the *y* only into *t-ty*, yielding *ty-ty*. Or, equivalently, using `\patterns{tt1y gg1y}` with context-sensitive ligatures changing *tt-y* to *ty-ty* and *gg-y* to *gy-gy*, etc. This solution uses up many character positions from the font, and many many extra ligatures are involved. Also, the user must know that to produce an actual *t-ty* (which almost never appears in Hungarian), `t{}-ty` must be used.

All of this can be accomplished using virtual fonts. The author has tested to see that the concept works by decompiling aer10.vf to aer10.vpl and modifying the `(LIGTABLE)`. However, automation of the virtual font generation is work remaining for the future.

Hyphenation of the double two-character consonants *ggy* and *ssz* is similar to *tty*. However, compound words such as *leggyakoribb* ("most frequent") and *vasszekér* ("iron chariot") should be hyphenated at the subword boundary without the addition of extra letters, i.e. as *leg-gyakoribb* and *vas-szekér*. Extra `\patterns` may be added, for example `\patterns{gg1y .leg1g4yakoribb.}`, to disable insertion of *y* for each important compound word. This is quite straightforward, because it does not require more ligatures (apart from the context sensitive ligature program changing *gg-y* to *gy-gy*).

One might suggest context-sensitive ligatures could be avoided if *ty* is introduced as a new single letter. But this won't work because, step (1): 'ttу' has to be converted to 'ty ty' using more than one ligature, and then step (2): further conversion to 't ty' if there is no line break, but TeX won't run its hyphenation algorithm in the middle of ligature processing, between steps (1) and (2).

The current approach of magyar.ldf for handling long double consonants is a compromise. By default, the patterns do not hyphenate those consonants, and the character ` is made active (with the standard Babel command `\declare@shorthand`), so that, for example `` `tty `` emits `t\nobreak \discretionary{y-}{}{}ty \nobreak \hskip \z@skip`. The first `\nobreak` is used to enable automatic hyphenation before the `` `tty `` construct, and the last `\nobreak` plus the `\hskip` enable hyphenation after the construct. Thus the word typed as `` megho`sszabbít `` will be hyphenated as `meg-hosz-szab-bít`. Similar shorthands are added for the other long consonants. The compromise is that the user has to be aware that he has to insert `s manually. A Perl script named ccs_extract.pl was developed to collect all occurrences of double consonants in a document so the user can review them and decide about the insertion of ` for each.

2.7 Table and Figure Captions

The document class defines `\@makecaption`, which is responsible for typesetting a caption for tables and figures. Some `.ldf` files, including frenchb.ldf and magyar.ldf override the default behaviour. magyar.ldf changes the colon separating the caption heading (e.g. "1. táblázat", or "Table 1") from the caption text to a full stop, in keeping with Hungarian typography. Furthermore, the `longcaption=` load option controls what should happen when the caption doesn't fit in a single line: whether it should

[2] `(LABEL C t) (LIG C - C y)` in the `.pl` file.

be centered, and whether there should be a line break after the caption heading.

The `tablecaptions=` and `figurecaptions=` load options control the appearance of the caption heading by redefining both `\fnum@table` and `\fnum@figure`. The default in both cases is to follow Hungarian typography, which requires the number to precede the table name.

2.8 Between the Section Title and the Section Number

The default definitions of `\@ssect` and `\@sect` separate the section number from the section title with a `\quad`. In Hungarian typopgraphy, only an `\enskip` is needed, and a dot has to be inserted after the number. The old version of magyar.ldf changed `\@sect` etc., but this caused conflicts with the AMS document classes and other packages, so that strategy has been given up in the new version of magyar.ldf. Instead, dots were moved into the `\numberline`, which adds the dot to `\tableofcontents` lines (being careful not to append the dot twice, see dot stripping code in section 2.26), and to `\@seccntformat`, which adds the dot and `\enskip` to the titles. The AMS document classes do not use `\@seccntformat`, so the AMS-specific `\tocsection` and `\tocappendix` commands had to be modified.

`\@numberline` also adds a dot after table and figure numbers in the `\listoftables` and `\listoffigures`, but the dot is needed there, too, anyway.

All three TOCs share a common problem related to language changes. Each time the language is changed, Babel emits changing commands to the three TOC files, so when they are re-read, each line comes in its appropriate language. The implementation has a flaw, however, because the `\write` to the TOC files gets executed when the page is shipped out, and the order of `\write`s on the same page follows the document structure: `\write`s in top insertions precede and bottom insertions follow those in the main text. So when a table or figure is moved to the top of the page, the writing of its TOC entry, together with the `\selectlanguage` command emitted by Babel is moved away, so `\selectlanguage` commands in the TOC files are reordered, which is wrong. The solution is to emit a `\selectlanguage` command for each TOC entry, so the TOC entries can be freely reordered. magyar.ldf implements this solution as a local fix, but it should be fixed generally in a new version of Babel.

2.9 Spacing Around Punctuation

It is quite easy to add extra space *after* punctuation characters with `\sfcode` (see “space factor” in chapter 13 of [3]). The LaTeX `\nonfrenchspacing` command (which is activated by default) assigns a space factor of 3000 to `.`, `?` and `!`, 2000 to `:`, 1500 to `;`, and 1250 to `,`.

However, adding extra space *before* punctuation needs a different approach. Both frenchb.ldf and magyar.ldf make the characters `:`, `;`, `!` and `?` active with the Babel `\initiate@active@char` interface, and insert unbreakable space in horizontal mode (`\ifhmode`) just before the punctuation character. This feature of magyar.ldf can be turned off using the `activespace=` load option, partly because making these four common characters active may lead to incompatibility with other packages, and partly because the extra space before punctuation is very rare in current Hungarian documents. In French typography, about one-third of a normal space is required before punctuation, and if it is not possible to add that amount with the typesetting technology, one full space should be added. However, in Hungarian, the fallback strategy is to omit the extra space.

The last action the active punctuation character should do is insert itself, but typing it verbatim into the definition will lead to an infinite loop. For example, `\catcode`?=13 \def?{\kern.1em ?}` will loop infinitely. The solution is to use `\string?` in place of the last `?`, so its catcode will be changed to 12 (other). Using `\edef` with this approach will make the macro a little bit faster, because `\string` will be executed only once, at load time.

2.10 Quoted Material

In English, text can be quoted using ‘single’ or “double” quotation marks. These can be nested into each other both ways. Hungarian provides three nesting levels: „outer »middle ’inner’«”. Although the guillemet symbols are missing from the CM fonts with OT1 encoding, this is not a serious problem, since Babel provides them (using the `\ll` relation: $\ll$ and $\gg$), and Hungarian text should be typeset with the T1 font encoding anyway, to allow hyphenation of words with accented characters.

frenchb.ldf provides `\LasyGuillemets` and `\CyrillicGuillemets` so the user can select the origin of the replacement guillemets. magyar.ldf relies on the defaults provided by Babel in the hope that the T1 encoding is used, so replacements are not needed. magyar.ldf doesn’t adjust spacing around the quotation symbols, but provides a `\textqq` command which emits quotations with proper nesting and spacing. For example, `\textqq{outer \textqq{middle \textqq{inner}\,}\,}` gives the above three-level sample. `\textqq` does English quotations (with two alternating levels) when the Hungarian language is not active.

2.11 List Environments

The default spacing, indentation and label item generation of list environments (such as `itemize` and `description`) are incorrect for Hungarian. The `labelenums=` and `labelitems=` load options control whether labels are modified to match Hungarian traditions. Five levels of depth are provided for both `itemize` and `enumerate`. The maximum depth is hardwired to the LaTeX definitions of these environments, so the `\ifnum\@enumdepth>3` test had to be changed to

```
\expandafter \ifx\csname \labelenum
  \romannumeral\the\@enumdepth\endcsname\relax
```

(and similarly for `\@itemdepth`).

Although the vertical space that the standard document classes leave around lists is too large, and the indentation is also incorrect, these problems have not yet been solved in magyar.ldf. (frenchb.ldf modifies `\itemsep` and other spacing dimensions to match French typographical rules.)

2.12 Modularity Using Load Options

The user can customize `.ldf`s using Babel's language attribute facility. For example, greek.ldf has `\bbl@declare@ttribute{greek} {polutoniko} {..}`, and so if the user loads greek.ldf with

```
\usepackage[greek]{babel}
\languageattribute{greek}{polutoniko}
```

the code in the `..` is run when `\languageattribute` is called. If present, `\languageattribute` must be part of the document preamble, and the `.ldf` file must be already loaded.

The fundamental problem with language attributes is that the user can pass only declared keywords, and not arbitrary data to the `.ldf` file, and — since attributes are processed too late — they cannot be used to control which parts of the `.ldf` files should be loaded.

Thus, magyar.ldf follows a different approach. Options are ⟨key⟩=⟨value⟩ pairs, which can be declared any time before magyar.ldf is loaded. The set of keys is fixed, but values can be arbitrary. It is the responsibility of the macro belonging to the key to verify that the syntax of the value is correct. None of the other `.ldf` files provide load option support this flexible. This option-passing scheme is similar to keyval.sty, but magyar.ldf doesn't actually use keyval.sty, because of a general design policy to avoid dependencies.

Since `.ldf` file names are the LaTeX options to babel.sty in the `\usepackage[···]{babel}` line, it is not possible to pass options to the individual `.ldf` files directly. However, LaTeX provides the command `\PassOptionsToPackage`, which declares options for a package before the package is loaded. So for example, `\PassOptionsToPackage{a=b,c=d}{foo.bar}` appends `a=b,c=d` to the macro `\opt@foo.bar`. magyar.ldf examines `\opt@magyar.ldf`, so for example passing options with `\PassOptionsToPackage{titles=\enskip}{magyar.ldf}` forces the space in section headings between the section number and the section title to be `\enskip`. (For compatibility reasons, magyar.ldf also processes the contents of `\magyarOptions` as options. This is useful to make magyar.ldf work with plain TeX.)

TeX macro wizards may enjoy studying the option parsing code in magyar.ldf. The entry point of the routine is `\processOptions`, whose argument is a comma-separated option list of ⟨key⟩=⟨value⟩ pairs. The routine calls `\processOption{`⟨key⟩`}{`⟨value⟩`}` for each pair found. This code is shown in figure 1.

2.13 Default Option Sets

Since there are 51 load options in the current magyar.ldf, the user should not be forced to know all of them. Reasonable defaults are provided (namely, `defaults=over-1.4`), so novice users can simply proceed. Intermediate users can select one of the five defaults, and possibly change a few options they don't like in their preferred default, and only expert users will change many options individually.

The number of bytes loaded were measured in a recent version of magyar.ldf, totalling 177 353 bytes. Out of that 177 kB, 32 417 bytes were used for initialization and providing the load option support framework, and declaring the options for the five defaults. After that, 138 872 bytes were used for implementing features selected by the load options. In the descriptions below, the number of feature bytes skipped is listed. (The larger the number, the less of magyar.ldf is processed at load time.)

The default sets are:

1. `=over-1.4` (10 720 bytes not loaded) This is the default among the defaults. Its main goal is to make all documents with the previous version of magyar.ldf (1.4) compile with the new version and to provide emergency bugfixes to incompatibility problems caused by the old version. It introduces a few essential typographical changes which have little impact on line

```
\def\processOptions#1{\processOptions@a#1,\hfuzz,}
\def\processOptions@a#1,{%
  \if,\noexpand#1,% (1)
    \expandafter\processOptions@a
  \else
    \csname fi\endcsname% (2)
    \processOptions@b#1,=,%
  \fi}
\@gobble\iftrue
\def\processOptions@b#1#2=#3,#4\fi{% (3) (4)
  \ifx\relax#4\relax
    \ifx#1\hfuzz% Terminator
      \expandafter\expandafter\expandafter
        \@gobble% (5)
    \else
      \if,\noexpand#1% OnlySpace
      \else% MissingArg; #2 ends by comma
        \if=\noexpand#1\missingKey#2%
        \else          \missingVal#1#2\fi
      \fi
    \fi
  \else% Normal
    \processOption{#1#2}{#3}%
  \fi
  \processOptions@a}
\def\missingKey#1,{\errmessage{Key missing
                     for value: #1}}
\def\missingVal#1,{\errmessage{Value (=) missing
                     for option: #1}}
\def\processOption#1#2{\typeout{Got option
                     key=(#1) val=(#2)}}
```

Figure 1: Option parsing code. Comments: (1) ignores extra commas, detects them by testing whether `#1` is empty; (2) this is a `\fi` when expanded, but doesn't count as a `\fi` when being skipped over because its surrounding condition is false. The real `\fi` won't be expanded, because it is parsed as the parameter terminator of `\processOptions@b`. (3) needs `#1#2` instead of just `#1`, so TeX will ignore space tokens in front of `#1`. As a side effect, when the option begins with `=`, the `=` will be put into `#1`, so `\missingKey` can be reported. (4) There are four different cases in which `\processOptions@b` can be invoked. The exact case is determined by how the macro parameter text separates parameters. The cases are: *Normal case*: `#1#2` is the key, `#3` is the value, `#4` is `=,`; *MissingArg case* (=⟨value⟩ is missing, or ⟨key⟩ is missing, but =⟨value⟩ is present): `#1#2` is ⟨key⟩`,` or =⟨value⟩`,`, `#3` and `#4` are empty; *Terminator case*: `#1#2` is `\hfuzz`, `#3` and `#4` are empty; *OnlySpace case*: `#1` is `,`, `#2`, `#3` and `#4` are empty. (5) `\@gobble` removes the call of `\processOptions@a` at the end of the macro, so the iteration is finished.

and page breaks; it disables big, eye-catching changes. It makes most new commands available, but doesn't turn new features on.

2. `=compat-1.4` (82 890 bytes not loaded) Implements a strict compatibility mode with version 1.4 of magyar.ldf. Documents look about the same (exact match not guaranteed), even when the output is typographically incorrect. It does not define new commands such as `\told` or `\emitdate`.
3. `=safest` (124 679 bytes not loaded) Turns off all features, reverts to LaTeX and Babel defaults in every respect. It is useful only for debugging purposes: if a document doesn't compile, but it compiles with `defaults=safest`, individual options can be turned on one-by-one to see which is causing the compatibility problem.
4. `=prettiest` (1 221 bytes not loaded) Turns on all new features, and tries to follow Hungarian typography in the prettiest, most advanced way. It is possible that compatibility problems will arise with other packages, although the author is not currently aware of any.
5. `=hu-min` (1 317 bytes not loaded) Follows Hungarian typographical rules as closely as possible. Compliance is not complete, of course, because some aspects are not implemented; thus they are not covered by load options, and they cannot be controlled using defaults. If typographical rules allow choice (e.g. the first paragraph of a section may or may not be indented), the easiest and most compatible solution is chosen (e.g. accept the indentation defined by the document class).

2.14 Skipping Parts of the Input File

Since some parts of magyar.ldf can be disabled using load options, as we have seen, it is desirable to skip them completely. The easiest way of skipping part of TeX code is wrapping it into `\ifnum\MyFeature<1` …`\fi`. But this kind of skipping will consume hash memory for new control sequences skipped over, and it also requires that the skipped part is properly nested with respect to `\if`…s. magyar.ldf defines the following macro to do skipping without these flaws.

```
\@gobble\iftrue
\def\skiplong#1{\fi
  \bgroup% so ^} would close it
  \catcode\string`^13
  \lccode\string`~=\string`^
  \lowercase{\let~\fi}%
  \catcode\string`\\14
  % comment, save hash memory
```

```
  \catcode\string`$14
  \iffalse}
\@gobble\fi
```

Now, code can be skipped with the construct

```
\ifnum\MyFeature<1 \skiplong\fi
  ...
  \@gobble
{^}
```

`\lowercase` is needed in the implementation because `\let^\fi` does not work, since the catcode of `^` is already assigned to be superscript when the definition of `\skiplong` is read.

2.15 Detecting Digits for Definite Articles

`\ref`, `\pageref` and `\cite` generate numbers, which are often prefixed by the definite article in Hungarian. The construct 'the \ref{foo}; works fine in English, but the Hungarian definite article has two forms: *a* and *az*. *Az* must be used if the following words (as pronounced) starts with a vowel, and *a* must be used for consonants. So we need a macro that generates the definite article for numbers automatically. magyar.ldf contains the macro `\az`, which prefixes its argument by either `a~` or `az~`. This kind of macro is at present unique to magyar.ldf; other `.ldf` files apparently do not implement solutions for similar problems in other langages.

Unfortunately, `\az` is not expandable, because it redefines the meaning of several commands before processing its argument. `\az` works by half-expanding its argument (fully expanding, of course, `\ref`, `\pageref` and `\cite`), ignoring braces and most control sequences, non-digit and non-letter characters, changing `\romannumeral` to `\number` (so that *x* will become *az x*, but `\az{\romannumeral␣10}` yields *a x*), looking for a number, a word or a single letter in the beginning (in fact, *az* has to be emitted if the starting digit is *5*, and *a* has to be emitted if the starting digit is not *1* or *5*). For words, single letters and positive numbers not beginning with *1*, the proper definite article depends on the first letter only.

For numbers starting with *1*, the definite article must be *az* if and only if the number of digits is $3k + 1$ for an integer k. For example, the Hungarian words for 1, 12, 123, 1000 are *egy*, *tizenkettő*, *százhuszonhárom*, *ezer*, respectively, and the definite forms are *az 1*, *a 12*, *a 123*, and *az 1000*, respectively. So we have to count the number of digits of a number.

It is not necessary to have 10 `\if` commands to test whether macro argument `#1` is a digit: it almost always works fine to use `\ifnum1<1\string#1␣`. The space at the end is important, because it will terminate the second number of the `\ifnum` if `#1` is a digit. The condition $(1 < 1)$ is false if `#1` is a non-digit, and true $(1 < 10,\ 1 < 11$ etc.) otherwise. `\string` cancels the special catcode `#1` might have. `#1` shouldn't be longer than a single token, because the test makes TeX process the extra tokens when `#1` contains a digit followed by extra tokens. If `#1` is `\if` or similar, the test isn't skippable. The test works even if `#1` is empty.

The macro `\Az` is also defined to insert a capitalized definite article at the beginning of a sentence. The macros `\aref`, `\Aref`, `\apageref`, `\Apageref`, `\acite` and `\Acite` are combinations of `\az` and referencing commands, so for example `\aref{foo}` is equivalent to `\az{\ref{foo}}`.

2.16 Counting Digits with Multiple Sentinels

A "sentinel" is something placed at the end of a list so that a conditional iteration over the list stops at the sentinel. For example, section 2.12 uses `\hfuzz` as a sentinel for the option processing of the macro `\processOptions@b`. A sentinel is usually a single token, but sometimes multiple sentinels have to be used in a row, when a macro processing them takes multiple parameters.

As mentioned in section 2.15, the Hungarian definite article (*a/az*) for a number depends on its pronounciation. The rule is: *az* has to be emitted for numbers starting with *5*, and for numbers starting with *1* and having the number of digits following *1* divisible by 3. All other numbers are preceded by *a*. magyar.ldf thus contains a macro that counts number of digits following it:

```
\def\digitthree#1{\digitthree@#1///\hbox$}
\def\digitthree@#1#2#3{%
  \csname digitthree@%
  \ifnum9<1\string#1 \ifnum9<1\string#2
    \ifnum9<1\string#3 %
  \else b\fi\else b\fi\else z\fi\endcsname}
\def\digitthree@z#1\hbox${z}
\def\digitthree@b#1\hbox${}
\message{1:\digitthree{} 100:\digitthree{23+}
  1000:\digitthree{456}}
```

In this example the macro `\digitthree` expands to *z* if its argument starts with digits of the multiple of 3. `\hbox$` is used as a sentinel to skip everything after the last digit has been found. The sentinel must not be present in the parameter itself. `\hbox$` makes no sense in TeX, so it is quite reasonable to assume that the parameter doesn't contain this. magyar.ldf uses `\hfuzz` and `\vfuzz` when only a single token is allowed, because these two unexpandable tokens are quite rare. The three consecutive slashes in the example are three sentinels, so `\digitthree@` has always enough arguments.

However, the test doesn't work if the parameter of `\digitthree` contains braces. For example `\digitthree{1{2x}}` would look for the undefined control sequence `\digitthree@x`.

In the example the `\csname` trick was used to avoid `\expandafter` in the nested `\if`s. See the definition of `\@@magyar@az@set` in magyar.ldf for using three multi-character sentinels in the same macro.

2.17 Definite Articles Before Roman Numerals

`\az` in magyar.ldf works differently for `\az{x}` and `\az{\romannumeral 10}` (see section 2.15), but how should it distinguish when `\romannumeral` has already been expanded by the time `\az` is called? Although there is no general solution to the problem, magyar.ldf addresses the case when `\az{\ref{my-part}}` is called, having the label `my-part` point to a `\part`, when `\def\thepart{\@Roman\c@part}` is active. (This is so with the standard book.cls.) `\ref` gets the part number from the `\newlabel{my-part}{{x}{42}}` command written to the `.aux` in the previous run of LaTeX. `\label`, which has emitted this `\newlabel` has already expanded `\romannumeral` in the previous run, long before our `\ref` is called.

To make the definite article work in this special case, magyar.ldf redefines `\label` so it writes `\hunnewlabel` in addition to `\newlabel` to the `.aux` file. The arguments of `\hunnewlabel` are pre-expanded when `\let\romannumeral\number` is in effect. This solution also works when `\pageref` refers to a roman numeral page number.

Expanding the page number at the right time is rather tricky. The LaTeX `\protected@write` says `\let\thepage\relax`, which prevents expansion in the following `\edef`, so `\thepage` is expanded only when the page is shipped out, and `\c@page` contains the right page number. What we want is to half-expand `\thepage`, so it gets expanded to `\@roman \c@page`, and `\@roman` is expanded to `\number` (!), but the expansion of `\number\c@page` is postponed until the page is shipped out. This can be done by defining `\def\romannumeral {\noexpand\number}` before calling `\protected@write`. In practice, magyar.ldf itself expands the page number, so three `\noexpand`s are needed in front of `\number`.

Redefining `\label` (so it emits `\hunnewlabel`) also raises a problem. Some packages loaded later might also override `\label`, for example hyperref.sty loads nameref.sty `\AtBeginDocument`, which overrides `\label`. magyar.ldf recognises the new definition when the Hungarian language is activated — which is done after the `\AtBeginDocument` hooks are run (see section 2.21). So `\hunnewlabel` works fine with hyperref.sty.

2.18 Removing All Braces

Removing all braces from a token list is required by the `\az` command (that inserts the *a/az* definite article). `\az` can find the first letter of its argument more easily if the argument doesn't contain braces.

The `\removebraces` macro defined in figure 2 removes all braces and spaces (recursively) from the tokens following it, until the first `\hfuzz`. The tokens may not contain a `\hfuzz` inside braces, but they may contain expandable material, even with unbalanced conditionals, because those are left unexpanded in `\removebraces@nobone` by `\noexpand`. The most important trick here is the construct `\ifcat{\noexpand#1`, which is true if `#1` starts with a brace, and yields `#1` with its first brace stripped. `\iffalse}\fi` is needed so that the macro definition is nested with respect to braces. The usage of `\@firstoftwo` is also worth mentioning: it is used to change the `\removebraces@nobone` token following the `\if` to `\removebraces`.

2.19 Changing `\catcode`s Safely

`\makeatletter` is equivalent to `\catcode 64 11` on ASCII systems; this changes the category code of characters having code 64 to 11 (letter). It is possible to specify the character `@` without knowing its character code: `` \catcode`@12 ``. Wherever TeX looks for a number (after `\catcode`, `\ifnum`, `\number`, etc.), it accepts a decimal number, an octal number prefixed by `'`, a hexadecimal number with digits `0-9A-F` prefixed by `"`, an internal counter (such as `\linepenalty`), a `\count` register (such as `\count42` or `\@listdepth`) or a character prefixed by `` ` ``. The character can be specified as a character token, or as a single-character control sequence. It is wise to specify `{`, `}`, `%` and space as `\{`, `\}`, `\%` and `\␣`, respectively, so the whole construct is properly nested with respect to braces, and since `%` and space tokens would be ignored.

However, many Babel language modules (`.ldf` files) make the character `` ` `` active (i.e. `\catcode` 13), so the definition of `` ` `` in `` \catcode`@12 `` gets expanded. The expansion can be prevented by using `\noexpand`, but `` \noexpand` `` yields `` ` ``$_{13}$, which is wrong, because `` ` ``$_{12}$ is needed, and moreover, will be expanded in the second run, because TeX is looking for a number. Fortunately, `` \string` `` solves the problem, because `\string` changes the `\catcode` of

```
\@gobble\iftrue
\def\removebraces@stop#1#2\fi{#1}%
\def\removebraces#1{\ifx\hfuzz#1\removebraces@stop\fi
  \expandafter\removebraces\expandafter{\ifcat{\noexpand#1\hfuzz\iffalse}\fi
  \expandafter\removebraces\else\hfuzz}\removebraces@nob{#1}\fi}
\def\removebraces@nob#1#2{#2\ifx\hfuzz#1\hfuzz\expandafter\@firstoftwo% #2 is \fi
  \expandafter\removebraces\fi\removebraces@nobone#1}
\def\removebraces@nobone#1{\noexpand#1\removebraces}
\message{R:\removebraces {{foo}{{}{b}{{{{a\fi}}}}r}}\hfuzz;}
```

Figure 2: `\removebraces`: A macro to remove all braces.

the following character token to 12 (other) or 10 (space); and, if a control sequence follows, `\string` converts it to a series of character tokens with `\catcode` other or space.

Thus, the ideal definition of `\makeatletter` is `\catcode\string`\@11␣`, which doesn't rely on the previous `\catcode` of ` or of `@`. The space at the end of the definition is needed so TEX knows that the number 11 won't be followed by subsequent digits. Of course, the definition works only when the characters `catcodestring` have `\catcode` letter, `\` is an escape character (`\catcode` 0), and space is a space (`\catcode` 10). These are reasonable assumptions, because none of the standard LATEX packages change them.

The LATEX kernel's definition of `\makeatletter` is `\catcode`\@11\relax`, having `\relax` instead of space, which is equally good to mark the end of a number. This definition doesn't need `\string`, because at the time it is read, the `\catcode` of ` is guaranteed to be 12 (other).

magyar.ldf saves the `\catcode` of `` ` ! & + - = | ; : ' " ? / `` in the beginning, changes them to other, and restores them just before `\endinput`. This is needed in case other `.ldf`s have been loaded (e.g. `\usepackage[french,magyar]{babel}`) that have redefined `\catcode`s. For example, french.ldf activates `! ? ; :`.

It is also good not to change `\catcode`s until `\begin{document}` (not even `\AtBeginDocument`), because other packages not yet loaded may depend on the old, unchanged `\catcode`s. Babel, unfortunately, activates a character immediately when a shorthand is defined in an `.ldf` file, so this can raise strange compatibility issues — which can be partly resolved by loading most other packages before Babel. magyar.ldf solves this by not touching the `\catcode` of its own shorthands at the time of definition, but instead calls `\bbl@activate` in `\extrasmagyar`, and `\bbl@deactivate` in `\noextrasmagyar`. This is a local and temporary solution only. Future versions of Babel are expected to postpone character activation as far as `\@preamblecmds` (see also section 2.21).

2.20 Shorthands

A shorthand is an active character defined by an `.ldf` file with the `\declare@shorthand` command provided by Babel. In this sense, all active punctuation characters (see section 2.9) are shorthands.

The most important shorthand in magyar.ldf is $`_{13}$. (Most `.ldf` files choose that character to be the main shorthand, but some, such as germanb.ldf, choose $"_{13}$.) The use of Hungarian shorthands can be disabled by the `active=` load option, and the shorthand character can be changed from ` with the `activeprefix=` load option. magyar.ldf also provides the ``\shu` `` command, which is a longer form of $`_{13}$, but without the possibly hazardous `\catcode` change.

Each shorthand is an active character, which raises compatibility problems (see section 2.19). magyar.ldf tries as hard as possible to avoid problems, but all efforts are in vain if another `.ldf` file is loaded which activates the same shorthand in the default (and unsafe) way.

For the user a shorthand is a control sequence without a backslash, so a shorthand is a command that can be typed and read quickly. germanb.sty provides `"a` to be equivalent to `\"a`, saving a keystroke for every accented German letter. magyar.ldf doesn't provide this saving, because the letters `o` and `u` have 3 accented forms, and introducing different letters for them would lead to confusion. Hungarian LATEX authors are encouranged to use the `latin2` encoding to type accented letters as a single character.

But the shorthand does an important job concerning (unaccented) long double consonants; for example, `` `tty `` is an abbreviation for `t\nobreak\discretionary{y-}{ }{ }ty\nobreak\hskip\z@skip`. (Section 2.6 explains why this is needed.) It should be noted that shorthands are implemented as TeX macros, so `␣{t}ty and `tty are equivalent.

The shorthand functionality of magyar.ldf for non-letters is inspired by ukraineb.ldf. `= and `- stand for a hyphen that separates words, so both words are automatically hyphenated by TeX (implemented as \leavevmode\nobreak-\hskip\z@skip); `- in math mode stands for a space character following a delimiter (\mskip2.4mu plus3.6mu minus1.8 mu) that will magically be exactly as wide as if a space was inserted outside math mode, because the implicit \mskip0.6mu after the delimiter is already subtracted; `-- emits \,--\,, to be used between author names in Hungarian bibliographies; `| emits a hyphen that is repeated at the beginning of the next line if the line is broken there (implementation: \leavevmode\nobreak-\discretionary {}{-}{-} \nobreak\hskip\z@skip), to be used with long words (e.g. *nátrium--klorid*) having important hyphens; `_ inserts a discretionary hyphen with automatic hyphenation enabled at both sides; `< inserts a French opening guillemet even if the ligature << is missing from the current font; `> inserts its paired closing; `" is equivalent to \allowbreak with hyphenation enabled on both sides (implementation: \hskip\z@skip); `~ inserts a hyphen that doesn't form ligatures when repeated (implementation: \textormath{\leavevmode\hbox{-}}{-}).

2.21 Inserting Code at \@preamblecmds

Babel calls \selectlanguage to set the default language \AtBeginDocument, which is (in general) too early. Suppose that the default language redefines \catcodes to be used with active characters. All packages that are loaded after the default language is activated will contain characters with unexpected and invalid catcodes. For example, if hyperref.sty is loaded after magyar.ldf, the \AtBeginDocument entries of hyperref.sty contain \RequirePackage{name ref}, which is executed *after* the entry \select language{magyar} of babel.sty, so nameref.sty will be loaded with wrong catcodes, and it will fail.

The solution is to postpone activation of the default language until after the \AtBeginDocument hooks. To accomplish this, magyar.ldf appends to \@preamblecmds, which is executed by the LaTeX kernel in \document, after \AtBeginDocument.

But what about the call to \selectlanguage inserted \AtBeginDocument by babel.def? Fortunately, it becomes a no-op, because magyar.ldf modifies \selectlanguage to do nothing if \languagename hasn't changed — and this is exactly the case when activating the default language. On the other hand, \@preamblecmds has to force the change even when \languagename is unchanged, so it calls \select@language (notice the at-sign). So magyar.ldf adds a call to \select@language to \@preamblecmds.

It also runs \pagestyle{headings} for the relevant document classes, so \ps@headings is executed once more, and the Hungarian version of the headings as defined by magyar.ldf will have a chance to be installed.

2.22 Displaying Theorem Titles

In English, theorem titles are displayed as "Theorem 1", but Hungarian requires "1. tétel.". To implement this, the \@begintheorem and \@opargbegin theorem macros are redefined each time the Hungarian language is activated. However, if theorem.sty or ntheorem.sty is loaded, the changes have to be embedded into a theorem style. The chosen name for the style is magyar-plain. It is activated by default when magyar.ldf is loaded, so theorem titles will come out right unless the user calls \theoremstyle. When amsthm.sty is loaded, magyar.ldf redefines the macros \thmhead and \swappedhead so both will emit the title properly.

2.23 Extra Symbols, \paragraph Titles, and Description Items

Hungarian typography requires a separator character other than a dot after the \paragraph title. Thus, a paragraph in English starting with "**title** text" should be something like "**title** ♦ text". magyar.ldf provides several pre-defined title separation symbols, selected by the load option postpara=; similarly, postsubpara= controls \subparagraphs and postdescription= controls \items in the description environment.

magyar.ldf redefines \paragraph in a truly ugly fashion when postpara= is active, so that no extra horizontal space is inserted after the title, but the title ends at the separation symbol. The default definition of \paragraph is based on \@startsection, whose argument #5 is a negative skip, which means a positive horizontal skip after the title. This is changed to -1sp by magyar.ldf to avoid the skip, and an optional argument is always passed to the original \paragraph so the title will be typeset with the separator.

2.24 Indentation after Section Titles

Hungarian typography allows the first paragraph after a section title to be either indented or unindented, so magyar.ldf provides afterindent= as the load option to control this. LaTeX calculates the value of a boolean variable \if@afterindent from

```
\expandafter\addto\csname
   \expandafter\ifx\csname mathoptions@on
    \endcsname\relax check@mathfonts
      \else mathoptions@on\fi
  \endcsname{\catcode`,12 \mathcode`,"8000
  \begingroup\lccode`~`,\lowercase
    {\endgroup\def~}{\HuComma}}
```

Figure 3: `\HuComma`: Smart commas in math.

the signedness of a parameter of `\@startsection`, and later uses that boolean to insert or omit the indentation. The value is forced to true by magyar.ldf by the simple definition `\let\@afterindentfalse \@afterindenttrue`.

2.25 The Decimal Comma

The dot character is defined as *ordinary* in mathematical text by default, so decimal real numbers can be typed simply as `$-12.34$`. Hungarian denotes the decimal point by a comma instead of a do, but typing `$-12,34$` yields '$-12, 34$' with too much space after the comma, because the comma is defined as *punctuation* rather *ordinary* in math text. `$-12{,}34$` yields '$-12{,}34$', which is correct, but magyar.ldf provides two mechanisms to save the two keystrokes of the curly braces around the comma.

First, the `\HuComma` macro below inserts an ordinary comma if it is followed by a digit, and an operator comma otherwise:

```
\edef\hucomma@lowa#1#2 #3#4 #5#6\hfuzz{%
  \noexpand\ifnum9<1#5 \noexpand\if#1t%
  \noexpand\if#3c% (1)
  \noexpand\mathord\noexpand\fi\noexpand\fi%
  \noexpand\fi\mathchar%
  \ifnum\mathcode`,="8000 "613B \else\the%
  \mathcode`, \space\fi}%
\def\hucomma@lowb{\expandafter\hucomma@lowa
  \meaning\reserved@a/ / /\hfuzz}%
\DeclareRobustCommand\HuComma
  {\futurelet\reserved@a\hucomma@lowb}
```

The solution Donald Arseneau proposed to the comma problem inspired these macros. In line (1) `\hucomma@lowa` tests whether the `\meaning` of the following character is 'the character ⟨digit⟩'. A `\meaning` is always at least three words, but it may be more (e.g. 'math shift character $'). Only *the character* starts with letters *t* and *c*. An `\edef` is needed above so the `\mathchar` emitted doesn't depend on `\mathcode` changes after the definition of `\HuComma`. Then the comma character can be redefined as `\HuComma`, as given in figure 3.

With these definitions, the formula '$F_i(x,y) = y^i + 1{,}3x,\ x,y \in A,\ i = 1,2,3,\ \ldots$' can be typed simply as `$F_{i}(x,y)=y^i+1,3x,\ x,y \in A, \ i=1,\ 2,\ 3,\ldots$`, if `\␣` is breakable (such as in nath.sty).

When nath.sty is loaded, the definitions are appended to `\mathoptions@on`, and if nath.sty is missing, to `\check@mathfonts`. The appropriate macro is run just before `\everymath` by LaTeX. Redefining the `\catcode` and `\mathcode` this way ensures that the proper comma is used inside math mode — unless the whole math formula is a macro argument with already assigned `\catcode`s. Also, it is not a good use of `\begingroup`, `\lccode`, `\lowercase` and `\endgroup` to modify the active meaning of a character without actually activating it. Calling `\catcode`,13` before `\def` wouldn't help here anyway if the whole construct is embedded into a macro definition, because `\catcode` wouldn't be able to change an already assigned catcode.

frenchb.ldf provides `\DecimalMathComma` and `\StandardMathComma` to change the `\mathcode` of the comma. However, the smart comma based on `\HuComma` acts correctly without the user needing to be aware of curly braces or redefinitions.

The solution above can be activated with the loading option `mathhucomma=fix`. An alternative approach doesn't alter `\mathcode`s, but introduces a special math mode in which the dot appears as a comma only when the Hungarian language is active. Thus the printout of `\MathReal{-12.34}` depends on the current Babel language. The definition of `\MathReal` in magyar.ldf is similar to:

```
\def\mathreal@lowa#1{\ensuremath{%
  \mathreal@lowb#1\@gobble.}}
\def\mathreal@lowb#1.{%
  #1\@secondoftwo\@gobble% (1)
  {\mathchar"013B \mathreal@lowb}}% comma
\DeclareRobustCommand\MathReal{\ensuremath}%
{\catcode`\ 11\relax\addto\extrasmagyar{%
\babel@save\MathReal %
\let\MathReal \mathreal@lowa}}
```

The argument of `\MathReal` must contain the dot to be changed literally, outside braces. There is a little macro wizardry in the implementation that stops calling `\mathreal@lowb` infinitely. The call `\mathreal@lowa` terminates its argument by a sentinel `\@gobble.`, so `#1` of `\mathreal@lowb` will end by `\@gobble`, which will gobble `\@secondoftwo`, so the `\@gobble` in line (1) will take effect, which stops the recursion.

`\MathReal` is going to be extended in the future so it will handle physical units following the number properly, and it will also insert thin spaces after each three digits. This feature has already been implemented in frenchb.ldf.

2.26 Parsing Dates

There are many correct ways to write dates in Hungarian, and magyar.ldf provides an `\emitdate` command that can generate any of these formats. Doing the reverse is a little more interesting.

Let's suppose we have a Gregorian date consisting of a year (4 or 2 digits), a month (a number or a name) and a day-of-month in some standard format. We want a command `\parsedate` to detect the format, split the date into fields, and call `\fixdate`:

```
\def\fixdate#1#2#3{%
  \@tempcnta#1 \ifnum#1<50
  \advance\@tempcnta2000 \fi
  \ifnum\@tempcnta<100
  \advance\@tempcnta1900 \fi
  \typeout{found year=(\@tempcnta)
  month=(#2) day=(#3)}}
```

Many dates have an optional dot at the end. Since that dot doesn't carry useful information, we should remove it first. The `\stripdot` command defined below expands to its argument with the trailing dot removed. `\stripdot` works only if the argument doesn't contain the token `\relax`. `\relax` is not special; any other token would have worked.

```
\def\stripdot#1{\expandafter%
  \stripdot@lowb\stripdot@lowa
  #1\relax.\relax}%
\def\stripdot@lowa#1.\relax{#1\relax}%
\def\stripdot@lowb#1\relax#2\relax{#1}%
```

The definition of `\parsedate` is shown in figure 4. `\parsedate` first does some generic cleanup, and puts the resulting date into `\re@b`. `\endgroup` cancels the redefinition of `\today` etc., but `\re@b` is expanded first, which defines itself, so the value of `\re@b` will be retained after `\endgroup`. After that, the trailing dot is stripped, and then various `\parsedate@···` commands are run. If a command recognises the date format, it puts a call to `\fixdate` into `\re@a`, which will be called at the end of `\parsedate`. Strange strings like `!//:!\hfuzz` are sentinels.

The idiom `\expandafter\endgroup\re@b` is an important trick for expanding a macro before the current group completes (and changes are undone). It usually contains definitions of other control sequences whose meanings are about to be retained after the end of the group. An alternative would be to inject such a definition using `\aftergroup`, but that only accepts a single token, so it would be very painful to make a macro definition with spaces and braces survive this way.

The individual `\parsedate@···` commands are given in figure 5. This implementation of date parsing isn't error-proof. If something weird is passed to `\parsedate`, it may produce surprising TeX errors. However, `\parsedate` can distinguish between different formats of correct input.

```
\def\parsedate#1{%
 \begingroup
   \def\today{\the\year-\the\month-\the\day}%ISO
   \let\protect\string
   % remove accents from Hungarian month names:
   \let\'\@firstofone
   \let~\space%change ‘2003.~okt’ to ‘2003. okt’
   \edef\re@b{\def\noexpand\re@b{#1}}%
 \expandafter\endgroup\re@b
 \edef\re@b{\expandafter\stripdot\expandafter
             {\re@b}}%
 \let\re@a\@empty \expandafter\parsedate@a\re@b
   !--!\hfuzz
 \ifx\re@a\@empty \expandafter\parsedate@f\re@b
   !//:!\hfuzz \fi
 \ifx\re@a\@empty \expandafter\parsedate@b\re@b
   !//!\hfuzz \fi
 \ifx\re@a\@empty \expandafter\parsedate@c\re@b
   !..!\hfuzz \fi
 \ifx\re@a\@empty \expandafter\parsedate@d\re@b
   !. xyz !\hfuzz \fi
 \ifx\re@a\@empty \expandafter\parsedate@e\re@b
   !xyz , !\hfuzz \fi
 \ifx\re@a\@empty \errmessage{Unrecognised date:
                              \re@b}%
 \else \re@a% call \fixdate
 \fi}
```

Figure 4: `\parsedate`: Parse date formats.

2.27 Setting Up French Spacing

Hungarian typography requires `\frenchspacing` to be turned on, but most LaTeX users fail to follow this requirement. Babel provides the command `\bbl@frenchspacing`, which turns French spacing on if it was off. The `frenchspacing=` load option of magyar.ldf controls how Hungarian text should behave. For the sake of symmetry, magyar.ldf provides `\@@magyar@antifrenchspacing`, which — contrary to the typographical requirement — turns french spacing off:

```
\def\@@magyar@antifrenchspacing{%
  \ifnum\the\sfcode‘\.=\@m
    \nonfrenchspacing
    \let\@@magyar@nonfrenchspacing%
      \frenchspacing
  \else \let\@@magyar@nonfrenchspacing\relax
  \fi}
\let\@@magyar@@nonantifrenchspacing%
  \frenchspacing
\addto\extrasmagyar{\@@magyar%
  @antifrenchspacing}
\addto\noextrasmagyar{\@@magyar%
```

```
\def\parsedate@a#1-#2-#3!#4\hfuzz{% ISO date: YYYY-MM-DD
  \ifx\hfuzz#4\hfuzz\else \ifnum1<1\string#1\relax
  \ifnum1<1\string#2\relax \ifnum1<1\string#3\relax
  \def\re@a{\fixdate{#1}{#2}{#3}}%
  \fi\fi\fi\fi}
\def\parsedate@b#1/#2/#3!#4\hfuzz{% LaTeX date: YYYY/MM/DD
  \ifx\hfuzz#4\hfuzz\else \ifnum1<1\string#1\relax
  \ifnum1<1\string#2\relax \ifnum1<1\string#3\relax
  \def\re@a{\fixdate{#1}{#2}{#3}}%
  \fi\fi\fi\fi}
\def\parsedate@c#1.#2.#3!#4\hfuzz{% English date: YYYY.DD.MM
  \ifx\hfuzz#4\hfuzz\else \ifnum1<1\string#1\relax
  \ifnum1<1\string#2\relax \ifnum1<1\string#3\relax
  \def\re@a{\fixdate{#1}{#3}{#2}}%
  \fi\fi\fi\fi}
% vvv English and Hungarian month names
\def\mon@jan{1} \def\mon@feb{2} \def\mon@mar{3} \def\mon@apr{4}
\def\mon@maj{5} \def\mon@may{5} \def\mon@jun{6} \def\mon@jul{7}
\def\mon@aug{8} \def\mon@sze{9} \def\mon@sep{9} \def\mon@okt{10}
\def\mon@oct{10} \def\mon@nov{11} \def\mon@dec{12}
\def\parsedate@d#1. #2#3#4#5 #6!#7\hfuzz{% {2003. oktober 25}
  \ifx\hfuzz#7\hfuzz\else
    % now: {#1}=={2003}, {#2#3#4#5}=={oktober}, {#6}=={25}
    \ifnum1<1\string#1\relax \ifnum1<1\string#6\relax
      \lowercase{%
        \expandafter\ifx\csname mon@#2#3#4\endcsname\relax\else
        \edef\re@a{\noexpand\fixdate{\number#1}{%
          \csname mon@#2#3#4\endcsname}{\number#6}}\fi}%
  \fi\fi\fi}
\def\parsedate@e#1#2#3#4 #5, #6!#7\hfuzz{% {October 25, 2003}
  \ifx\hfuzz#7\hfuzz\else
    \ifnum1<1\string#5\relax \ifnum1<1\string#6\relax
      \lowercase{%
        \expandafter\ifx\csname mon@#1#2#3\endcsname\relax\else
        \edef\re@a{\noexpand\fixdate{\number#6}{%
          \csname mon@#1#2#3\endcsname}{\number#5}}\fi}%
  \fi\fi\fi}
\def\parsedate@f#1/#2/#3:#4!#5\hfuzz{% LaTeX default \today
  % YYYY/MM/DD:XX:YY
  \ifx\hfuzz#5\hfuzz\else \ifnum1<1\string#1\relax
  \ifnum1<1\string#2\relax \ifnum1<1\string#3\relax
  \def\re@a{\fixdate{#1}{#2}{#3}}%
  \fi\fi\fi\fi}
```

Figure 5: Individual `\parsedate`$\cdots$ commands.

@nonantifrenchspacing}

2.28 varioref.sty Fixes

The `magyar` load option of varioref.sty (2001/09/04 v1.3c) is buggy, because it uses the never-defined `\aza` command for adding definite articles, and it also calls `\AtBeginDocument` too late, producing a LaTeX error each time the Hungarian language is activated. magyar.ldf contains the correct definitions for the language-specific text reference macros, such as `\reftextlabelrange`, and also contains ugly fix-up code to remove the wrong macros inserted by varioref.sty. A patch has been sent recently to the author of varioref.sty.

Some of these text reference macros use the `\az` and the `\told` commands defined by magyar.ldf.

2.29 Removing Full Stops After Section Titles in AMS Classes

AMS document classes always append a full stop after section titles, which is strictly forbidden in Hungarian typography. The solution is to remove the tokens `\@addpunct.` from the definition of `\@sect` (and also from `\NR@sect` in case nameref.sty has also been loaded). But this simple idea is quite complicated to program, and the result is ugly, as seen in figure 6. This detects AMS classes by the presence of `\global\@nobreaktrue\@xsect` in the definition of `\@sect`, and adds code just before `\@xsect`. The code added prepends `\let\@addpunct\@gobble` to the definition of `\@svsechd`. `\@svsechd` is later called by `\@xsect`, which calls `\@addpunct`, but by that time `\@addpunct` is a no-op. The application of this fix is controlled by the `amspostsectiondot=` load option.

2.30 Reduced Math Skips

Investigations in [1] have shown that the following settings produce the desired space for Hungarian math mode:

```
\thickmuskip 4mu plus  2mu minus4mu
   % LaTeX: 5mu plus5mu
\medmuskip   2mu plus1.5mu minus2mu
   % LaTeX: 4mu plus2mu minus4mu
\thinmuskip  3mu % LaTeX: ditto
```

Notice that `\medmuskip` < `\thinmuskip`. These settings can be selected in magyar.ldf with the load option `mathmuskips=`. The difference between the original and the reduced spacing:

$$a + b - c/d * y \circ x = z \qquad a{+}b{-}c/d{*}y{\circ}x{=}z$$

2.31 Breaking a Long Inline Math Formula

Hungarian typography requires that a binary relation or operator (e.g. in $1 + 2 = 3 + 4$) must be repeated in the next line if an inline math formula is broken there. This can be accomplished for the equation sign by substituting `=\nobreak\discretionary{}{\hbox{\(=\)}}{}` for `=` in math formulas. The long inline formula delimiters `\(` and `\)` are used because the catcode of the `$` would be wrong if nath.sty was loaded after magyar.ldf. `\nobreak` is necessary, so TeX itself won't break the line after the $=$.

The `mathbrk=` load option of magyar.ldf controls whether the operators and relations should be redefined. If so, the operators $+$, $-$, $*$ (as well as 37 operators available as control sequences) and the relations $<$, $>$, $=$, $:$ (as well as 43 relations available as control sequences) are modified so they get repeated at the beginning of the line. The `\cdot` and the `\slash` operators are also modified, because Hungarian typography disallows breaking the line around them.

2.32 Restarting Footnote Numbering on Each Page

Although `\usepackage[perpage]{footmisc}` and footnpag.sty provide these features, magyar.ldf allows normal arabic footnote and page-restarting asterisk-footnotes to be intermixed. It is common in Hungarian article collections to have the notes of the author numbered in arabic (by `\footnote`), and the footnotes of the editor added with asterisks (by `\editorfootnote`). The first four editorial footnotes on a page are marked with *, **, ***, and †. magyar.ldf also inserts proper additional space between the footnote mark and the footnote text, and the footnote facility is fully customizable with the `\footnotestyle` command.

The basic idea behind the implementation of pagewise numbering is creating a `\label` for each footnote, and whenever the `\pageref` for that label shows a different page, resetting the counter to zero. This clobbering can be automated by abusing the `\cl@footnote` hook. Each time `\stepcounter` advances a counter, the corresponding `\cl@...` hook is called, which usually resets other counters (for example, advancing the chapter counter resets the section counter). But arbitrary code can be executed after the automatic `\stepcounter{footnote}` by appending that code to the macro `\cl@footnote`.

The famous problem of creating a macro that will expand to n asterisks is proposed in appendix D of *The TeXbook* [3]. David Kastrup has provided

```
\expandafter\amssect@fixa\@sect[][][][][][][][]%
  \global\@nobreaktrue\@xsect\hfuzz\@sect
\expandafter\amssect@fixa\@sect[][][][][][][][]%
  \global\@nobreaktrue\@xsect\hfuzz\NR@sect% with nameref.sty
\long\def\amssect@fixa#1\global\@nobreaktrue\@xsect#2\hfuzz#3{%
  \ifx\hfuzz#2\hfuzz\else\amssect@fixb#3\fi}% fix if found
\def\amssect@fixb#1{% #1 is \@sect or \NR@sect
  \expandafter\let\csname amssect@saved\string#1\endcsname#1%
  \edef#1{\noexpand\expandafter\noexpand\amssect@low\expandafter
    \noexpand\csname amssect@saved\string#1\endcsname}%
  \let\@svsechd\@empty% prevent Undefined \cs
  \long\def\amssect@low##1\global\@nobreaktrue{##1%
    \expandafter\def\expandafter\@svsechd\expandafter{%
      \expandafter\let\expandafter\@addpunct\expandafter\@gobble
      \@svsechd}%
    \global\@nobreaktrue}}
```

Figure 6: Removing full stops after AMS section titles.

a brilliant solution to the problem in [2], namely `\expandafter\mtostar\romannumeral\number`n`000A`, where `\mtostar` transforms `m`s to asterisks: `\def\mtostar#1{\if#1m*\expandafter\mtostar\fi}`. This solution is used in magyar.ldf.

magyar.ldf also provides the following command to insert footnotes into section titles such that neither the table of contents nor the page headings are affected:

```
\def\headingfootnote{%
  \ifx\protect\@typeset@protect%
    \expandafter\footnote
  \else\expandafter\@gobble\fi
```

2.33 Class-specific Modifications

magyar.ldf does some modifications based on the current document class (using the `\@ifclassloaded` LaTeX command). Only the standard classes article.cls, report.cls, book.cls and letter.cls are supported at present. The visual appearance of the `\part` and `\chapter` output is changed, and the page headers are also modified. For book.cls, part numbering is spelled out, so "Part 1" becomes "Első rész" ("Part One") if the load option `partnumber=` is set to `Huordinal`.

The command `\ps@headings` has to be executed again to install its changed heading macros. This is called from `\@preamblecmds`, after the default language has been activated (see section 2.21).

The typographically correct customization of letter.cls is under development.

2.34 Spelling Out Numerals and Ordinals

The `\@hunumeral` and `\@huordinal` macros defined in magyar.ldf can spell out integers between -9999 and 9999. `\@Hunumeral` and `\@Huordinal` are the capitalized versions of these macros. For example, `\@huordinal{2004}` produces *kétezer-negyedik* ("two thousand and fourth") and `\@Hunumeral{2004}` produces *Kétezer-négy* ("Two thousand and four"). All of these macros are fully expandable, so they can be used for `\part` numbering: `\def\thepart{\@Huordinal\c@part}`, or more simply: `\def\thepart {\Huordinal{part}}`.

The most important implementation issue is the method to retrieve the last digit of a number in an expandable construct. If the number is between 0 and 9999, the following macro solves the problem:

```
\def\LastDigitOf#1{\expandafter%
   \lastdigit@a\number#1;}%
\def\lastdigit@a#1;{% #1 in 0..9999
  \ifnum#1<10 #1\else\ifnum#1<100
    \lastdigit@b00#1%
  \else\ifnum#1<1000 \lastdigit@b0#1%
    \else\lastdigit@b#1\fi
\def\lastdigit@b#1#2#3#4{#4}
```

2.35 Suffix Generation

As mentioned earlier, the Hungarian language has suffixes to represent relations in space and time, instead of prepositions. For example, an English math text might contain "It follows from (1)", in which "from (1)" can be typed as `from (\ref{eq1})`. The LaTeX referencing scheme guarantees that the text above will come out right, even if the order of equations is changed in the document.

But in Hungarian, the suffix standing in place of "from" has two forms: *-ból/-ből*, depending on the vowel harmony of the pronounciation of `\ref{eq1}`. So there is a need for automatic suffix generation.

magyar.ldf provides the command `\told`, with which the Hungarian version of "from (1)" can be typed as `\told(\ref{eq1})+bol{}`, which will generate "(1)-ből", "(2)-ből", but "(3)-ból".

`\told` can handle 20 different suffixes, and $4 \cdot 20$ suffix combinations (such as `\told3+adik+ra{}`, meaning "to the third"). Only the last number is considered in references containing multiple numbers. Roman numerals are recognised properly in references with the help of `\hunnewlabel` (see section 2.17 — this is implemented similar to `\az`). Suffix generation is supported only for integers and Hungarian document structure names (see section 3.2), because writing a generic suffix generator without a database is quite a difficult task, and definitely won't give Hungarian LaTeX users much more comfort beyond the current `\told` implementation.

Although most Hungarian suffixes have 1, 2 or 3 forms,[3] numbers can be classified into 23 paradigm classes, so that the paradigm class uniquely determines the correct form of all known suffixes. The reason that there are so many classes is because the letter *v* of the *-val/-vel* suffix must be also changed to the last letter of the number if that letter is a consonant. Essentially each final digit has a class, and there are classes for the powers of 10, and some of the numbers $20, 30, \ldots 90$ also have their own classes. To sum up, the suffix of a number depends on the last nonzero digit, and the number of trailing zeroes.

The implementation of `\told` is suprisingly long and ugly, full of recursive macros that parse the input, and it doesn't contain any bright ideas that are not also found elsewhere in magyar.ldf. The curious TeX hacker should study `\az` instead, because it is shorter and its trick density is much higher.

2.36 Warning Messages

magyar.ldf has the unique feature that it displays warning messages (called 'suggestions') at load time to notify the user that they are using magyar.ldf in a possibly incorrect way. If they are not disabled by the `suggestions=` load option, the following suggestions are displayed to standard output during the `\AtBeginDocument` hook:

- the user forgot to load `\usepackage{t1enc}` — so words with accented letters won't hyphenate automatically;
- the user forgot to load `\usepackage[latin2]{inputenc}`, or the input encoding chosen is not `latin2`, `cp1250` or `utf8` — so there is a good chance that accented characters will disappear or come out wrong;
- the Hungarian hyphenation patterns requested were not found — magyar.ldf tries to use the other two possible Hungarian patterns, if they are available;
- `\def\magyarOptions` or `\PassOptionsToPackage{...}{magyar.ldf}` was specified too late — late options can be detected, but they have no effect, since options do their work while magyar.ldf is being *loaded*;
- the buggy varioref.sty has been loaded as `\usepackage[magyar]{varioref}` — this will happen until the patch is integrated to varioref.sty; the current version is so buggy that it displays an untraceable LaTeX error each time the Hungarian language is activated (see section 2.28).

[3] Of course, suffixes with only one form are not supported by `\told`.

3 Miscellaneous Tricks

First we show some common expansion tool macros defined by LaTeX:

```
\def\@empty{}
\long\def\@gobble#1{}
\long\def\@gobbletwo#1#2{}
\long\def\@firstofone#1{#1}
\long\def\@firstoftwo#1#2{#1}
\long\def\@secondoftwo#1#2{#2}
```

`\@firstofone` differs from `\@empty`, because it may not be followed by `}`, it ignores spaces in front of its argument, and it removes at most one pair of braces around its argument. All of these properties are consequences of the macro expansion rules described in chapter 20 of *The TeXbook* [3].

This remainder of this section describes TeX macro and typesetting tricks not tightly related to the Hungarian language.

3.1 The Factorial Sign in Math Mode

nath.sty contains a smart definition of the factorial operator, so $(a + b)!/a!\,b! + c! \cdot d!$, with proper spacing can be typed as `$(a+b){!}/a!b!+c!\cdot d!$`. The only place where braces are needed is before the slash. magyar.ldf adapts the definition:

```
\def\factorial{\mathchar"5021\mathopen{}%
  \mathinner{}}
\expandafter\addto\csname \expandafter\ifx
  \csname mathoptions@on\endcsname\relax
  % detect nath.sty
  check@mathfonts\else mathoptions@on\fi
   \endcsname{\catcode`!12
     \mathcode`!"8000
  \begingroup\lccode`~`!\lowercase{%
   \endgroup\def~}{\factorial}}
```

```
\def\NumbersToOne#1{\nonumbers@a#1\hfuzz}
\def\nonumbers@skipa#1\nonumbers@s{#1\nonumbers@a}
\def\nonumbers@a#1{% change first digit
  \ifx#1\hfuzz \expandafter\@gobble
  \else\if1<1\string#11\else\if\noexpand#1mi%
  \else\if\noexpand#1di\else\if\noexpand#1ci%
  \else\if\noexpand#1li\else\if\noexpand#1xi%
  \else\if\noexpand#1vi\else\if\noexpand#1ii%
  \else\if\noexpand#1MI\else\if\noexpand#1DI%
  \else\if\noexpand#1CI\else\if\noexpand#1LI%
  \else\if\noexpand#1XI\else\if\noexpand#1VI%
  \else\if\noexpand#1II%
  \else\noexpand#1\nonumbers@skipa
  \fi\fi\fi\fi\fi\fi\fi \fi\fi\fi\fi
  \fi\fi\fi\fi\fi \nonumbers@s}
\def\nonumbers@s#1{% gobble next digits
  \ifx#1\hfuzz \expandafter\@gobble
  \else\if1<1\string#1\else\if\noexpand#1m%
  \else\if\noexpand#1d\else\if\noexpand#1c%
  \else\if\noexpand#1l\else\if\noexpand#1x%
  \else\if\noexpand#1v\else\if\noexpand#1i%
  \else\if\noexpand#1M\else\if\noexpand#1D%
  \else\if\noexpand#1C\else\if\noexpand#1L%
  \else\if\noexpand#1X\else\if\noexpand#1V%
  \else\if\noexpand#1I%
  \else\noexpand#1\nonumbers@skipa
  \fi\fi\fi\fi\fi\fi\fi \fi\fi\fi\fi
  \fi\fi\fi\fi\fi \nonumbers@s}
```

Figure 7: `\nonumbers`: Change all roman and arabic numbers to 1.

3.2 Including the Structure Name in References

Text like "in subsection 5.6" is usually typed as `in subsection~\ref{that}`. But it would be nice if LaTeX were able to guess that `\ref{that}` actually points to a subsection. The structure depth can be deduced by counting the dots in the reference: a subsection has one dot (in an article), and a subsubsection has two dots.

magyar.ldf provides the `\refstruc` command which has a smarter detection scheme: it changes all roman and arabic numbers to one (`1`, `i` or `I`) in the reference, and compares the result with the tokens generated by `\thechapter`, `\thesection` etc., with `\c@chapter` etc. set to 1 temporarily. This should work in most cases, although it cannot refer to equations, tables or figures. In Hungarian text, the Hungarian structure names are emitted, and other texts the original, English control sequence names are printed. `\refstruc` includes the definite article and suffixes support; for example,

`\Az{\refstruc{that+tol}}`

may emit *az 1. fejezettől* ("from chapter 1").

The full implementation is quite long, and is not included here, but the macro that changes all roman and arabic numbers to one is presented in figure 7.

```
\ifnum 0<%
  \expandafter\ifx\csname setTrue\endcsname
    \relax\else1\fi
  \expandafter\ifx\csname allowttyhyphens\endcsname
    \relax\else1\fi
\space
  \def\amsfix#1#2#3#4#5#6#7\vfuzz{%
    \def\reserved@a{#6}%
    \ifx\@tocline#2\ifx\reserved@a\@empty%
      \def#1{\@tocline{#3}{#4}{#5}{\parindent}{}}%
    \fi\fi}
  \expandafter\amsfix\expandafter\l@table \l@table
       ,,,,,,,\vfuzz
  \expandafter\amsfix\expandafter\l@figure\l@figure
       ,,,,,,,\vfuzz
\fi
```

Figure 8: Fixing overfull `\hbox`es in AMS classes.

3.3 Enabling Long Page Numbers

If the width of a page number in the table of contents is greater than `\@pnumwidth`, LaTeX emits an "Overfull `\hbox`" warning. This can be eliminated by changing `\@dottedtocline` in the LaTeX kernel. The line `\hb@xt@\@pnumwidth{\hfil \normalfont \normalcolor #5}` should be changed to:

```
\setbox\@tempboxa\hbox{\normalfont
R \normalcolor #5}%
\ifdim\wd\@tempboxa<\@pnumwidth\setbox%
 \@tempboxa\hb@xt@\@pnumwidth{\hfil\unhbox
 \@tempboxa}\fi  \box\@tempboxa
```

Although this change isn't related to the Hungarian language, magyar.ldf will do it given the `dottedtocline=` load option.

3.4 Removing AMS Warnings from `\listoftables`

Some AMS document classes (such as amsart.cls) produce an "Overfull `\hbox`" warning for each line in the `\listoftables` and `\listoffigure`. This can be fixed by changing this line in the `\l@table` and `\l@figure` macros in the AMS classes:

`\@tocline{0}{3pt plus2pt}{0pt}{}{}{}`

to:

`\@tocline{0}{3pt plus2pt} {0pt}{}{\parindent}{}`.

The code shown in figure 8 makes this change.

The control sequences `\allowttyhyphens` and `\setTrue` are defined by each of the AMS document classes, so their presence indicates that one of those classes are loaded. The logical or operation using `\ifx`s nested to the `\ifnum` test is also worth noting.

3.5 Discarding to End of File

The naïve solution `\ifskiprest\endinput¶\fi` results in the TeX error message "`\end` occurred when `\iftrue` in line n was incomplete" if there is a line break at the ¶ sign. Without the line break, the naïve solution works perfectly, because `\endinput` stops reading the current file *after* the current line, so the `\fi` also gets evaluated.

It is possible to do something before `\endinput`:

```
\expandafter\ifx\csname
  ver@foo.sty\endcsname\relax
  \endinput \errmessage{I am incompatible
    with foo.sty}\fi
```

All of the above must be put after `\endinput` without a line break. The `\csname fi\endcsname` construct closes the `\ifx` when the condition is true, but is invisible when the condition is false, and TeX is skipping tokens.

The LaTeX kernel macro `\@ifpackageloaded` implements the conditional end by a different trick. The following two constructs are equivalent:

```
\@ifpackageloaded{foo}{\endinput\errmessage{I
  am incompatible with foo.sty}}{}
```

and

```
\expandafter\ifx\csname ver@foo.sty\endcsname
      \relax\expandafter\@gobble
\else \expandafter\@firstoftwo \fi
{\endinput
\errmessage{I am incompatible with foo.sty}}
```

In the trick above, `\errmessage` isn't on the same line as `\endinput`. This isn't a problem, because by the time `\endinput` is evaluated by TeX's stomach, `\errmessage` has been read from the file, and it is already on the input stack. `\endinput` doesn't discard the input stack, it just prevents more lines from being read from the current file.

3.6 Typesetting Text Verbatim

`\catcode`s are assigned when TeX's eyes read the next character from the current line, so a `\catcode` command affects all subsequent characters of the current line, as well as the following lines. But once a category code has been assigned, it won't be affected by subsequent `\catcode` commands. For example, `` \@firstofone{\catcode`A 14 AAA} `` makes `A` into a comment start character, but it emits three `A`s, because the category code of the three `A`s is already set by the time `\catcode` is executed.

The reason why the `\verb` command of the LaTeX kernel cannot be part of a macro argument is the same: `\verb` changes the `\catcode` of most characters to *other*, but these changes have no effect inside a macro argument, because the argument has been read from the input file by the time `\catcode` can take effect.

Instead of altering `\catcode`s, a verbatim macro can be based on the `\meaning` primitive, so that it can be passed as an argument. However, TeX's eyes will have destroyed some information such as comments and the exact number of successive spaces by the time `\meaning` is expanded. For example, these definitions are from binhex.dtx:

```
\def\verbatize#1{\begingroup
  \toks0{#1}\edef\next{\the\toks0}%
  \dimen0\the\fontdimen2\font
             \fontdimen2\font=0pt
  \expandafter\stripit \meaning\next
  \fontdimen2\font=\dimen0 \endgroup}
\def\stripit#1>{}
```

3.7 Stopping the Iteration

Let's suppose we need a macro that capitalizes all as and bs until the first ".":

```
\def\ucab#1{%
  \if\noexpand#1.\expandafter\@gobble
  \else\if\noexpand#1aA%
  \else\if\noexpand#1bB%
  \else\noexpand#1%
  \fi\fi\fi\ucab}
\message{\ucab abc.abc} % -> ABcabc
```

`\noexpand` prevents expansion of `#1` in case it is an expandable control sequence such as `\the` or a macro. If `\if` is changed to `\ifx`, then not only the character codes, but also the category codes would be compared.

The trick that stops the iteration here is that `\expandafter` expands the first `\else` that will remove everything up to the last `\ucab`. Then comes `\@gobble`, which removes `\ucab`, and the iteration is stopped.

The construct doesn't work when `#1` has braces around it, or it is `\if...`, `\else` or `\fi`. Also, spaces will be ignored because of macro expansion.

But what if we'd like to capitalize only the *first* a or b? Then we would need `\expandafter\expandafter\expandafter\@gobble` after A, and seven `\expandafter`s after B. But `\expandafter` can be completely avoided using a different approach, based on macro arguments:

```
\def\helpif#1#2{#1\@firstoftwo}
\def\ucabs#1{%
   \if\noexpand#1.\helpif\fi\@secondoftwo{}
  {\if\noexpand#1a\helpif\fi\@secondoftwo{A}
  {\if\noexpand#1b\helpif\fi\@secondoftwo{B}
  {\noexpand#1\ucabs}}}%
}\message{\ucabs cbbas} % -> cBbas
```

It is not possible to move `\fi` into the definition of `\helpif`, because then TeX won't see that particular `\fi` when it is skipping the whole `\if` ... `\helpif` construction. With a small rearrangement we can get rid of `\@secondoftwo`:

```
\@gobble{\iftrue\iftrue}
   % \def\helpjf... contains 2*\fi
\def\helpjf#1\fi{#1\expandafter\@firstoftwo
  \else\expandafter\@secondoftwo\fi}
\def\ucabj#1{%
   \helpjf\if\noexpand#1.\fi{}
  {\helpjf\if\noexpand#1a\fi{A}
  {\helpjf\if\noexpand#1b\fi{B}
  {\noexpand#1\ucabs}}}%
}\message{\ucabj cdabs} % -> cdAbs
```

The line containing `\@gobble` above is needed so that `\def\helpjf` can be put inside `\iffalse` ... `\fi`, and TEX's `\fi` count won't decrease when seeing the two `\fi` tokens in the definition of `\helpjf`.

3.8 Appending Tokens to a Macro

TEX doesn't provide primitives for modifying the expansion text of a macro, but it is possible to define a new macro with the contents of the old one and some additional tokens. For example, `\expandafter\def \expandafter\foo\expandafter{\foo\iffalse$}` appends the two tokens `\iffalse$` to the macro `\foo`. Three `\expandafter`s were used to make the old `\foo` only expanded once. None of the above tokens were expanded, fortunately. Any tokens can be appended this way, as long as they are nested with respect to braces. But care has to be taken when doubling `#`s:

```
\def\AppendTo#1#2{\expandafter\def\expandafter
  #1\expandafter{#1#2}}
\def\foo{} \AppendTo\foo{#}
```

yields the TEX error "Illegal parameter number in definition of `\foo`". This can be solved by using token list registers, which double their hashmarks when expanded in an `\edef`:

```
\def\AppendTo#1#2{\begingroup
  \expandafter\toks\expandafter0%
  \expandafter{\foo#2}%
  \global\edef#1{\the\toks0}\endgroup}
\def\foo{} \AppendTo\foo{#x}
\show\foo % \foo=macro:-> ##x
```

Note that when a macro is defined, `#`s have to be properly formulated, and it is `\def` which eats half of `#`s (and converts `#1` etc. to special, inaccessible tokens), but `\edef` doesn't convert $\#_6$ to special tokens if it comes from a token list register. The disadvantage of the second definition of `\AppendTo` is that it must be `\global`. (There is a much longer solution that manually doubles the `#`s.) The `\addto` command of Babel and the `\vref@addto` command of varioref.sty are `\global`, similar to this solution.

3.9 Processing Arbitrary Package Options

LATEX packages can use the standard commands `\DeclareOption`, `\ExecuteOption` and `\ProcessOptions` to access package options passed to them, and these commands work fine with a fixed set of options. The `\DeclareOption*` command can be used to declare arbitrary options:

```
%\DeclareOption{10pt}{\typeout{got ten-pt}}%(1)
\DeclareOption*{\typeout{got=(\CurrentOption)}}
\ProcessOptions  % in file foo.sty
```

Two lines will be printed when `foo.sty` is loaded as `\usepackage[,foo=bar,,no,]{foo}`. These are `got=(foo=bar)` and `got=no`. The optional argument of `\usepackage` may contain spaces and/or a single newline around commas and at the ends. Class options are passed to `\DeclareOption*`, so when `\documentclass[10pt]{article}` is active, `got=(10pt)` will not appear, but when line (1) is uncommented, `got ten-pt` will appear.

There is an alternative, low-level way for accessing all the options at once:

```
\AtEndOfPackage{\let\@unprocessedoptions
  \relax}% prevent warning
\typeout{\csname opt@\@currname.\@currext
  \endcsname}
```

This prints the full option list with extra spaces and newlines removed, but commas, including superfluous ones, are kept intact.

4 Beyond the Current magyar.ldf

4.1 Other Hungarian Typesetting Software

Although magyar.ldf contains most of the functionality needed for following Hungarian typographic traditions, other utilities and packages can help in typesetting Hungarian texts. Most of this software, including magyar.ldf, is going to be available under the name MagyarLATEX from `http://www.math.bme.hu/latex/`.

magyar.ldf The new Hungarian module for Babel. Version 1.5 was written by Péter Szabó beginning in the autumn of 2003.

huhyph.tex **or** huhyph3.tex The old (version 3) Hungarian hyphenation `\patterns` for the T1 encoding, written by Gyula Mayer in 1998. Part of most TEX distributions. See section 2.1.

huhyphc.tex The new version of the Hungarian hyphenation `\patterns` for the T1 encoding, written by Gyula Mayer in 2002 [4]. Part of most TEX distributions. Hyphenates foreign compound words on the subword boundary, e.g. *szin-kron*. See section 2.1.

huhyphf.tex The new version of the Hungarian hyphenation `\patterns` for the T1 encoding, written by Gyula Mayer in 2002 r[4]. Part of most TEX distributions. Hyphenates foreign compound words phonetically, e.g. *szink-ron*. See section 2.1.

ccs_extract.pl A Perl script that helps with hyphenation of words containing long double Hungarian consonants. It finds all occurrences of such words in the document and lets the user decide whether to insert, for each unique word, the magyar.ldf shorthands for `\discretionary` breaks for long double consonants. The program was written by Péter Szabó in 2003.

lafmtgen.pl An easy-to-use Perl script that can generate format files (`.fmt`) containing all hyphenation patterns required by the specified LATEX document. It has some other features related to generating and installing format files, and is to be used with the Unix teTEX distribution. It was written by Péter Szabó in 2003.

huplain.bst BIBTEX style file for Hungarian bibliographies, based on plain.bst. It encourages sharing the same `.bib` database between Hungarian and English documents. It follows the (simple) convention that the style of a bibliography depends on the language of the document containing the entry, not the language of the entry itself. It was written by Péter Szabó in 2003.

husort.pl A drop-in replacement of makeindex that follows the Hungarian standards of index sorting and typesetting. It is implemented as a Perl script. It was written by Péter Szabó in 2003.

magyar.xdy Hungarian style file for the Xîndy index processing program. Implements the Hungarian sorting order and a Hungarian typesetting style. The implemented sorting order does not follow Hungarian rules as strictly and elegantly as husort.pl. It was written by Péter Szabó in 2003.

CM-Super The EC fonts in Type 1 format in T1 and various other encodings. It is not part of MagyarLATEX, but is available from CTAN. It is useful for converting Hungarian text to PDF, so the generated PDF file will contain the EC fonts in Type 1 format, and will be rendered quickly and nicely by Acrobat Reader.

MagyarISpell The Hungarian language database of the Ispell spell checker for Unix. On Debian systems, it can be installed with the command `apt-get install ihungarian`.

Ispell has a TEX mode, which skips control sequences and comments when checking TEX source. (Unfortunately, the arguments of `\begin{tabular}` and many other non-textual elements of LATEX documents are not skipped.) Ispell can be used interactively, but this method is not comfortable, and incremental checking is not possible.

Ispell also has an interprocess communication protocol, through which it can be integrated into text editors. For example, Emacs has built-in Ispell support to mark incorrect words visually. OpenOffice, LyX, editors in KDE and newer versions of Vim can do the same. MagyarISpell works fine in these editors. It is not part of MagyarLATEX, but it is freely available.

Note, however, that both the database and the stemmer of MagyarISpell is far from perfect, but among the Hungarian spell checkers only this one works inside Ispell, so only this can be easily integrated into editors.

MSpell A commercial Hungarian spell checker with a no-cost Linux download, developed by Morphologic (a company in Hungary that produces linguistic software). Doesn't have an interactive mode, but can replace Ispell in inter-process communication mode. A shell script is provided that replaces the `ispell` command, so MSpell can be integrated into text editors more easily. It is not part of MagyarLATEX.

HunSpell The successor of MagyarISpell, but based on a different spell checking architecture. It understands Hungarian much better than MagyarISpell, but since it is not based on Ispell, it is harder to integrate into text editors. For example, it is not available from the Emacs spell checking menu, even if it is installed. It is not part of MagyarLATEX, but it is freely available.

4.2 Future Work

Some features are still missing from magyar.ldf:

- letter.cls is not customized properly, the left indentation of the nested list environments is also not customized;
- a macro to emit numbers with groups of three digits separated is missing;
- layout.sty and many other packages don't have Hungarian captions yet;
- the shorthand ‘ is not disabled in math mode to give nath.sty a chance to typeset H_{symm} with `$H_{‘symm}$`;
- `\hunnewlabel` should store `table`, `figure` or `equation`, so `\refstruc` can insert it;
- new fonts and/or methods should be developed in place of `‘tty`;

- Hungarian typography needs a baseline grid, which is almost impossible to enforce in LaTeX;
- some of the separation symbols proposed for after `\paragraph` are not available yet;
- section titles should not be larger than normal text;
- bold fonts should be substituted for bold extended fonts, whenever available — and never with an error or warning message; page numbers should be removed from blank pages;
- the width of `\parindent` should be computed based on `\textwidth`;
- the length of the last line of a paragraph should not be too near to the right margin, especially if `\parindent` = 0;
- `\vskip`s above and below sections should be reduced;
- providing the commands `\H` and `\.` for OT1-encoded typewriter fonts;
- `\MathReal` should be extended with physical units;
- virtual fonts to support `\umlautlow` in T1 encoding;
- bold in `\begin{description}` should be `\emph`;
- allow specifying some compile-time options at run-time;
- `\textqq` should also work as an environment;
- `\told` should generate suffixes for month names.

Other features should be implemented outside magyar.ldf, as external programs. All programs in section 4.1 need improvement in one way or another.

4.3 Conclusion

An updated magyar.ldf which closely follows Hungarian typographical rules and works together with the most popular LaTeX packages without problems, has been awaited for many years. This new version is ready, as a single file longer than anything before, and it is filled with many advanced features.

Most of the features adapt LaTeX to Hungarian typographical rules, but some of them are bug fixes to various external packages, including design flaws and compatibility issues in Babel itself.

The implementation of some features clearly shows that TeX macro programming is an obscure and ineffective way of solving some of the language-related problems. It is hoped that new versions of Ω, together with the new version of Babel, will provide a framework in which such problems can be addressed compactly and elegantly, without constant awareness of actual and possible compatibility glitches.

References

[1] Gyöngyi Bujdosó and Ferenc Wettl. On the localization of TeX in Hungary. *TUGBoat*, 23(1):21–26, 2002.

[2] David Kastrup. De ore leonis. Macro expansion for virtuosi. In *EuroBachoTeX*, May 2002.

[3] Donald E. Knuth. *The TeXbook*. Addison-Wesley, 1984.

[4] Gyula Mayer. The Hungarian hyphenation module of TeX and LaTeX. Unpublished article in Hungarian, 10 July 2002.

[5] Petr Sojka. Notes on compound word hyphenation in TeX. *TUGboat*, 16(3):290–296, 1995.