NAME

hunspell − format of Hunspell dictionaries and affix files

DESCRIPTION

Hunspell(1) requires two files to define the language that it is spell checking. The first file is a dictionary containing words for the language, and the second is an "affix" file that defines the meaning of special flags in the dictionary.

A dictionary file (*.dic) contains a list of words, one per line. The first line of the dictionaries (except personal dictionaries) contains the approximate word count (for optimal hash memory size). Each word may optionally be followed by a slash ("/") and one or more flags, which represents affixes or special attributes. Dictionary words can contain also slashes with the "" syntax. Default flag format is a single (usually alphabetic) character. After the dictionary words there are also optional fields separated by tabulators or spaces (spaces only work as morphological field separators, if they are followed by morphological field ids, see also Optional data fields).

Personal dictionaries are simple word lists. Asterisk at the first character position signs prohibition. A second word separated by a slash sets the affixation.

```
foo
Foo/Simpson
*bar
```

In this example, "foo" and "Foo" are personal words, plus Foo will be recognized with affixes of Simpson (Foo's etc.) and bar is a forbidden word.

An affix file (*.aff) may contain a lot of optional attributes. For example, **SET** is used for setting the character encodings of affixes and dictionary files. **TRY** sets the change characters for suggestions. **REP** sets a replacement table for multiple character corrections in suggestion mode. **PFX** and **SFX** defines prefix and suffix classes named with affix flags.

The following affix file example defines UTF-8 character encoding. 'TRY' suggestions differ from the bad word with an English letter or an apostrophe. With these REP definitions, Hunspell can suggest the right word form, when the misspelled word contains f instead of ph and vice versa.

```
SET UTF-8
TRY esianrtolcdugmphbyfvkwzESIANRTOLCDUGMPHBYFVKWZ'

REP 2
REP f ph
REP ph f

PFX A Y 1
PFX A 0 re .

SFX B Y 2
SFX B 0 ed [^y]
SFX B y ied y
```

There are two affix classes in the dictionary. Class A defines a 're-' prefix. Class B defines two '-ed' suffixes. First suffix can be added to a word if the last character of the word isn't 'y'. Second suffix can be added to the words terminated with an 'y'. (See later.) The following dictionary file uses these affix classes.

```
3
hello
try/B
work/AB
```

All accepted words with this dictionary: "hello", "try", "tried", "work", "worked", "rework", "reworked".

GENERAL OPTIONS

Hunspell source distribution contains more than 80 examples for option usage.

SET encoding

Set character encoding of words and morphemes in affix and dictionary files. Possible values: UTF-8, ISO8859−1 − ISO8859−10, ISO8859−13 − ISO8859−15, KOI8-R, KOI8-U, microsoft-cp1251, ISCII-DEVANAGARI.

FLAG value

Set flag type. Default type is the extended ASCII (8-bit) character. 'UTF-8' parameter sets UTF-8 encoded Unicode character flags. The 'long' value sets the double extended ASCII character flag type, the 'num' sets the decimal number flag type. Decimal flags numbered from 1 to 65000, and in flag fields are separated by comma. BUG: UTF-8 flag type doesn't work on ARM platform.

COMPLEXPREFIXES

Set twofold prefix stripping (but single suffix stripping) for agglutinative languages with right-to-left writing system.

LANG langcode

Set language code. In Hunspell may be language specific codes enabled by LANG code. At present there are az_AZ, hu_HU, TR_tr specific codes in Hunspell (see the source code).

IGNORE characters

Ignore characters from dictionary words, affixes and input words. Useful for optional characters, as Arabic diacritical marks (Harakat).

AF number_of_flag_vector_aliases

AF flag_vector

Hunspell can substitute affix flag sets with ordinal numbers in affix rules (alias compression, see makealias tool). First example with alias compression:

```
3
hello
try/1
work/2
```

AF definitions in the affix file:

```
SET UTF-8
TRY esianrtolcdugmphbyfvkwzESIANRTOLCDUGMPHBYFVKWZ'
AF 2
AF A
AF AB
```

It is equivalent of the following dic file:

```
3
hello
try/A
work/AB
```

See also tests/alias* examples of the source distribution.

Note: If affix file contains the FLAG parameter, define it before the AF definitions.

Note II: Use makealias utility in Hunspell distribution to compress aff and dic files.

AM number_of_morphological_aliases

AM morphological_fields

Hunspell can substitute also morphological data with ordinal numbers in affix rules (alias compression). See tests/alias* examples.

OPTIONS FOR SUGGESTION

Suggestion parameters can optimize the default n-gram, character swap and deletion suggestions of Hunspell. REP is suggested to fix the typical and especially bad language specific bugs, because the REP suggestions have the highest priority in the suggestion list. PHONE is for languages with not pronunciation based orthography.

KEY characters_separated_by_vertical_line_optionally

Hunspell searches and suggests words with one different character replaced by a neighbor KEY character. Not neighbor characters in KEY string separated by vertical line characters. Suggested KEY parameters for QWERTY and Dvorak keyboard layouts:

KEY qwertyuiop|asdfghjkl|zxcvbnm
KEY pyfgcrl|aeouidhtns|qjkxbmwvz

Using the first QWERTY layout, Hunspell suggests "nude" and "node" for "*nide". A character may have more neighbors, too:

KEY qwertzuop|yxcvbnm|qaw|say|wse|dsx|sy|edr|fdc|dx|rft|gfv|fc|tgz|hgb|gv|zhu|jhn|hb|uji|kjm|jn|iko|lkm

TRY characters

Hunspell can suggest right word forms, when they differ from the bad input word by one TRY character. The parameter of TRY is case sensitive.

NOSUGGEST flag

Words signed with NOSUGGEST flag are not suggested. Proposed flag for vulgar and obscene words (see also SUBSTANDARD).

MAXNGRAMSUGS num

Set number of n-gram suggestions. Value 0 switches off the n-gram suggestions.

NOSPLITSUGS

Disable split-word suggestions.

SUGSWITHDOTS

Add dot(s) to suggestions, if input word terminates in dot(s). (Not for OpenOffice.org dictionaries, because OpenOffice.org has an automatic dot expansion mechanism.)

REP number_of_replacement_definitions

REP what replacement

We can define language-dependent phonetic information in the affix file (.aff) by a replacement table. First REP is the header of this table and one or more REP data line are following it. With this table, Hunspell can suggest the right forms for the typical faults of spelling when the incorrect form differs by more, than 1 letter from the right form. For example a possible English replacement table definition to handle misspelled consonants:

REP 8
REP f ph
REP ph f
REP f gh
REP gh f

```
REP j dg
REP dg j
REP k ch
REP ch k
```

Note I: It's very useful to define replacements for the most typical one-character mistakes, too: with REP you can add higher priority to a subset of the TRY suggestions (suggestion list begins with the REP suggestions).

Note II: Suggesting separated words by REP, you can specify a space with an underline:

```
REP 1
REP alot a_lot
```

Note III: Replacement table can be used for a stricter compound word checking (forbidding generated compound words, if they are also simple words with typical fault, see CHECKCOMPOUNDREP).

MAP number_of_map_definitions

MAP string_of_related_chars

We can define language-dependent information on characters that should be considered related (i.e. nearer than other chars not in the set) in the affix file (.aff) by a character map table. With this table, Hunspell can suggest the right forms for words, which incorrectly choose the wrong letter from a related set more than once in a word.

For example a possible mapping could be for the German umlauted ü versus the regular u; the word Frühstück really should be written with umlauted u's and not regular ones

```
MAP 1
MAP uü
```

PHONE number_of_phone_definitions

PHONE what replacement

PHONE uses a table-driven phonetic transcription algorithm borrowed from Aspell. It is useful for languages with not pronunciation based orthography. You can add a full alphabet conversion and other rules for conversion of special letter sequences. For detailed documentation see http://aspell.net/man-html/Phonetic-Code.html. Note: Multibyte UTF-8 characters have not worked with bracket expression yet. Dash expression has signed bytes and not UTF-8 characters yet.

OPTIONS FOR COMPOUNDING

BREAK number_of_break_definitions

BREAK character_or_character_sequence

Define new break points for breaking words and checking word parts separately. Use ^ and $ to delete characters at end and start of the word. Rationale: useful for compounding with joining character or strings (for example, hyphen in English and German or hyphen and n-dash in Hungarian). Dashes are often bad break points for tokenization, because compounds with dashes may contain not valid parts, too.) With BREAK, Hunspell can check both side of these compounds, breaking the words at dashes and n-dashes:

```
BREAK 2
BREAK -
BREAK --    # n-dash
```

Breaking are recursive, so foo-bar, bar-foo and foo-foo--bar-bar would be valid compounds. Note: The default word break of Hunspell is equivalent of the following BREAK definition:

BREAK 3
BREAK -
BREAK ^-
BREAK -$

Hunspell doesn't accept the "-word" and "word-" forms by this BREAK definition:

BREAK 1
BREAK -

W Note II: COMPOUNDRULE is better (or will be better) for handling dashes and other compound joining characters or character strings. Use BREAK, if you want check words with dashes or other joining characters and there is no time or possibility to describe precise compound rules with COMPOUNDRULE (COMPOUNDRULE has handled only the last suffixation of the compound word yet).

Note III: For command line spell checking of words with extra characters, set WORDCHARS parameters: WORDCHARS --- (see tests/break.*) example

COMPOUNDRULE number_of_compound_definitions

COMPOUNDRULE compound_pattern

Define custom compound patterns with a regex-like syntax. The first COMPOUNDRULE is a header with the number of the following COMPOUNDRULE definitions. Compound patterns consist compound flags, parentheses, star and question mark meta characters. A flag followed by a '*' matches a word sequence of 0 or more matches of words signed with this compound flag. A flag followed by a '?' matches a word sequence of 0 or 1 matches of a word signed with this compound flag. See tests/compound*.* examples.

Note: en_US dictionary of OpenOffice.org uses COMPOUNDRULE for ordinal number recognition (1st, 2nd, 11th, 12th, 22nd, 112th, 1000122nd etc.).

Note II: In the case of long and numerical flag types use only parenthesized flags: (1500)*(2000)?

Note III: COMPOUNDRULE flags haven't been compatible with the COMPOUNDFLAG, COMPOUNDBEGIN, etc. compound flags yet (use these flags on different words).

COMPOUNDMIN num

Minimum length of words in compound words. Default value is 3 letters.

COMPOUNDFLAG flag

Words signed with COMPOUNDFLAG may be in compound words (except when word shorter than COMPOUNDMIN). Affixes with COMPOUNDFLAG also permits compounding of affixed words.

COMPOUNDBEGIN flag

Words signed with COMPOUNDBEGIN (or with a signed affix) may be first elements in compound words.

COMPOUNDLAST flag

Words signed with COMPOUNDLAST (or with a signed affix) may be last elements in compound words.

COMPOUNDMIDDLE flag

Words signed with COMPOUNDMIDDLE (or with a signed affix) may be middle elements in compound words.

ONLYINCOMPOUND flag

Suffixes signed with ONLYINCOMPOUND flag may be only inside of compounds (Fuge-elements in German, fogemorphemes in Swedish). ONLYINCOMPOUND flag works also with words (see tests/onlyincompound.*).

COMPOUNDPERMITFLAG flag
: Prefixes are allowed at the beginning of compounds, suffixes are allowed at the end of compounds by default. Affixes with COMPOUNDPERMITFLAG may be inside of compounds.

COMPOUNDFORBIDFLAG flag
: Suffixes with this flag forbid compounding of the affixed word.

COMPOUNDROOT flag
: COMPOUNDROOT flag signs the compounds in the dictionary (Now it is used only in the Hungarian language specific code).

COMPOUNDWORDMAX number
: Set maximum word count in a compound word. (Default is unlimited.)

CHECKCOMPOUNDDUP
: Forbid word duplication in compounds (e.g. foofoo).

CHECKCOMPOUNDREP
: Forbid compounding, if the (usually bad) compound word may be a non compound word with a REP fault. Useful for languages with ‘compound friendly’ orthography.

CHECKCOMPOUNDCASE
: Forbid upper case characters at word bound in compounds.

CHECKCOMPOUNDTRIPLE
: Forbid compounding, if compound word contains triple repeating letters (e.g. foo|ox or xo|oof). Bug: missing multi-byte character support in UTF-8 encoding (works only for 7-bit ASCII characters).

SIMPLIFIEDTRIPLE
: Allow simplified 2-letter forms of the compounds forbidden by CHECKCOMPOUNDTRIPLE. It’s useful for Swedish and Norwegian (and for the old German orthography: Schiff|fahrt -> Schiffahrt).

CHECKCOMPOUNDPATTERN number_of_checkcompoundpattern_definitions

CHECKCOMPOUNDPATTERN endchars[/flag] beginchars[/flag] [replacement]
: Forbid compounding, if the first word in the compound ends with endchars, and next word begins with beginchars and (optionally) they have the requested flags. The optional replacement parameter allows simplified compound form. Note: COMPOUNDMIN doesn’t work correctly with the compound word alternation, so it may need to set COMPOUNDMIN to lower value.

COMPOUNDSYLLABLE max_syllable vowels
: Need for special compounding rules in Hungarian. First parameter is the maximum syllable number, that may be in a compound, if words in compounds are more than COMPOUNDWORDMAX. Second parameter is the list of vowels (for calculating syllables).

SYLLABLENUM flags
: Need for special compounding rules in Hungarian.

OPTIONS FOR AFFIX CREATION

PFX flag cross_product number

PFX flag stripping prefix [condition [morphological_fields...]]

SFX flag cross_product number

SFX flag stripping suffix [condition [morphological_fields...]]
: An affix is either a prefix or a suffix attached to root words to make other words. We can define affix classes with arbitrary number affix rules. Affix classes are signed with affix flags. The first line of an affix class definition is the header. The fields of an affix class header:

  (0) Option name (PFX or SFX)

(1) Flag (name of the affix class)

(2) Cross product (permission to combine prefixes and suffixes). Possible values: Y (yes) or N (no)

(3) Line count of the following rules.

Fields of an affix rules:

(0) Option name

(1) Flag

(2) stripping characters from beginning (at prefix rules) or end (at suffix rules) of the word

(3) affix (optionally with flags of continuation classes, separated by a slash)

(4) condition.

Zero stripping or affix are indicated by zero. Zero condition is indicated by dot. Condition is a simplified, regular expression-like pattern, which must be met before the affix can be applied. (Dot signs an arbitrary character. Characters in braces sign an arbitrary character from the character subset. Dash hasn't got special meaning, but circumflex (^) next the first brace sets the complementer character set.)

(5) Optional morphological fields separated by spaces or tabulators.

OTHER OPTIONS

CIRCUMFIX flag

Affixes signed with CIRCUMFIX flag may be on a word when this word also has a prefix with CIRCUMFIX flag and vice versa.

FORBIDDENWORD flag

This flag signs forbidden word form. Because affixed forms are also forbidden, we can subtract a subset from set of the accepted affixed and compound words.

FULLSTRIP

With FULLSTRIP, affix rules can strip full words, not only one less characters.

Note: conditions may be word length without FULLSTRIP, too.

KEEPCASE flag

Forbid uppercased and capitalized forms of words signed with KEEPCASE flags. Useful for special orthographies (measurements and currency often keep their case in uppercased texts) and writing systems (e.g. keeping lower case of IPA characters).

Note: With CHECKSHARPS declaration, words with sharp s and KEEPCASE flag may be capitalized and uppercased, but uppercased forms of these words may not contain sharp s, only SS. See germancompounding example in the tests directory of the Hunspell distribution.

Note: Using lot of zero affixes may have a big cost, because every zero affix is checked under affix analysis before the other affixes.

ICONV number_of_ICONV_definitions

ICONV pattern pattern2
: Define input conversion table.

OCONV number_of_OCONV_definitions

OCONV pattern pattern2
: Define output conversion table.

LEMMA_PRESENT flag
: Not used in Hunspell 1.2. Use "st:" field instead of LEMMA_PRESENT.

NEEDAFFIX flag
: This flag signs virtual stems in the dictionary. Only affixed forms of these words will be accepted by Hunspell. Except, if the dictionary word has a homonym or a zero affix. NEEDAFFIX works also with prefixes and prefix + suffix combinations (see tests/pseudoroot5.*).

PSEUDOROOT flag
: Deprecated. (Former name of the NEEDAFFIX option.)

SUBSTANDARD flag
: SUBSTANDARD flag signs affix rules and dictionary words (allomorphs) not used in morphological generation (and in suggestion in the future versions). See also NOSUGGEST.

WORDCHARS characters
: WORDCHARS extends tokenizer of Hunspell command line interface with additional word character. For example, dot, dash, n-dash, numbers, percent sign are word character in Hungarian.

CHECKSHARPS
: SS letter pair in uppercased (German) words may be upper case sharp s (ß). Hunspell can handle this special casing with the CHECKSHARPS declaration (see also KEEPCASE flag and tests/germancompounding example) in both spelling and suggestion.

Morphological analysis

Hunspell's dictionary items and affix rules may have optional space or tabulator separated morphological description fields, started with 3-character (two letters and a colon) field IDs:

```
word/flags po:noun is:nom
```

Example: We define a simple resource with morphological informations, a derivative suffix (ds:) and a part of speech category (po:):

Affix file:

```
SFX X Y 1
SFX X 0 able . ds:able
```

Dictionary file:

```
drink/X po:verb
```

Test file:

```
drink
drinkable
```

Test:

```
$ analyze test.aff test.dic test.txt
> drink
analyze(drink) = po:verb
stem(drink) = po:verb
> drinkable
analyze(drinkable) = po:verb ds:able
stem(drinkable) = drinkable
```

You can see in the example, that the analyzer concatenates the morphological fields in *item and arrangement* style.

Optional data fields

Default morphological and other IDs (used in suggestion, stemming and morphological generation):

ph: Alternative transliteration for better suggestion. It's useful for words with foreign pronunciation. (Dictionary based phonetic suggestion.) For example:

```
Marseille ph:maarsayl
```

st: Stem. Optional: default stem is the dictionary item in morphological analysis. Stem field is useful for virtual stems (dictionary words with NEEDAFFIX flag) and morphological exceptions instead of new, single used morphological rules.

```
feet  st:foot  is:plural
mice  st:mouse is:plural
teeth st:tooth is:plural
```

Word forms with multiple stems need multiple dictionary items:

```
lay po:verb st:lie is:past_2
lay po:verb is:present
lay po:noun
```

al: Allomorph(s). A dictionary item is the stem of its allomorphs. Morphological generation needs stem, allomorph and affix fields.

```
sing al:sang al:sung
sang st:sing
sung st:sing
```

po: Part of speech category.

ds: Derivational suffix(es). Stemming doesn't remove derivational suffixes. Morphological generation depends on the order of the suffix fields.

In affix rules:

```
SFX Y Y 1
SFX Y 0 ly . ds:ly_adj
```

In the dictionary:

ably st:able ds:ly_adj
able al:ably

is: Inflectional suffix(es). All inflectional suffixes are removed by stemming. Morphological generation depends on the order of the suffix fields.

feet st:foot is:plural

ts: Terminal suffix(es). Terminal suffix fields are inflectional suffix fields "removed" by additional (not terminal) suffixes.

Useful for zero morphemes and affixes removed by splitting rules.

work/D ts:present

SFX D Y 2
SFX D 0 ed . is:past_1
SFX D 0 ed . is:past_2

Typical example of the terminal suffix is the zero morpheme of the nominative case.

sp: Surface prefix. Temporary solution for adding prefixes to the stems and generated word forms. See tests/morph.* example.

pa: Parts of the compound words. Output fields of morphological analysis for stemming.

dp: Planned: derivational prefix.

ip: Planned: inflectional prefix.

tp: Planned: terminal prefix.

Twofold suffix stripping

Ispell's original algorithm strips only one suffix. Hunspell can strip another one yet (or a plus prefix in COMPLEXPREFIXES mode).

The twofold suffix stripping is a significant improvement in handling of immense number of suffixes, that characterize agglutinative languages.

A second 's' suffix (affix class Y) will be the continuation class of the suffix 'able' in the following example:

SFX Y Y 1
SFX Y 0 s .

SFX X Y 1
SFX X 0 able/Y .

Dictionary file:

```
drink/X
```

Test file:

```
drink
drinkable
drinkables
```

Test:

```
$ hunspell -m -d test <test.txt
drink st:drink
drinkable st:drink fl:X
drinkables st:drink fl:X fl:Y
```

Theoretically with the twofold suffix stripping needs only the square root of the number of suffix rules, compared with a Hunspell implementation. In our practice, we could have elaborated the Hungarian inflectional morphology with twofold suffix stripping.

Extended affix classes

Hunspell can handle more than 65000 affix classes. There are three new syntax for giving flags in affix and dictionary files.

FLAG long command sets 2-character flags:

```
FLAG long
SFX Y1 Y 1
SFX Y1 0 s 1
```

Dictionary record with the Y1, Z3, F? flags:

```
foo/Y1Z3F?
```

FLAG num command sets numerical flags separated by comma:

```
FLAG num
SFX 65000 Y 1
SFX 65000 0 s 1
```

Dictionary example:

```
foo/65000,12,2756
```

The third one is the Unicode character flags.

Homonyms

Hunspell's dictionary can contain repeating elements that are homonyms:

```
work/A    po:verb
work/B    po:noun
```

An affix file:

```
SFX A Y 1
SFX A 0 s . sf:sg3

SFX B Y 1
SFX B 0 s . is:plur
```

Test file:

```
works
```

Test:

```
$ hunspell -d test -m <testwords
work st:work po:verb is:sg3
work st:work po:noun is:plur
```

This feature also gives a way to forbid illegal prefix/suffix combinations.

Prefix--suffix dependencies

An interesting side-effect of multi-step stripping is, that the appropriate treatment of circumfixes now comes for free. For instance, in Hungarian, superlatives are formed by simultaneous prefixation of *leg-* and suffixation of *-bb* to the adjective base. A problem with the one-level architecture is that there is no way to render lexical licensing of particular prefixes and suffixes interdependent, and therefore incorrect forms are recognized as valid, i.e. **legvén = leg + vén* 'old'. Until the introduction of clusters, a special treatment of the superlative had to be hardwired in the earlier **HunSpell** code. This may have been legitimate for a single case, but in fact prefix--suffix dependences are ubiquitous in category-changing derivational patterns (cf. English *payable*, *non-payable* but **non-pay* or *drinkable*, *undrinkable* but **undrink*). In simple words, here, the prefix *un-* is legitimate only if the base *drink* is suffixed with *-able*. If both these patters are handled by on-line affix rules and affix rules are checked against the base only, there is no way to express this dependency and the system will necessarily over- or undergenerate.

In next example, suffix class R have got a prefix 'continuation' class (class P).

```
PFX P Y 1
PFX P   0 un . [prefix_un]+

SFX S Y 1
SFX S   0 s . +PL

SFX Q Y 1
SFX Q   0 s . +3SGV

SFX R Y 1
SFX R   0 able/PS . +DER_V_ADJ_ABLE
```

Dictionary:

```
2
drink/RQ        [verb]
drink/S  [noun]
```

Morphological analysis:

```
> drink
drink[verb]
drink[noun]
> drinks
drink[verb]+3SGV
drink[noun]+PL
> drinkable
drink[verb]+DER_V_ADJ_ABLE
> drinkables
drink[verb]+DER_V_ADJ_ABLE+PL
> undrinkable
[prefix_un]+drink[verb]+DER_V_ADJ_ABLE
> undrinkables
[prefix_un]+drink[verb]+DER_V_ADJ_ABLE+PL
> undrink
Unknown word.
> undrinks
Unknown word.
```

Circumfix

Conditional affixes implemented by a continuation class are not enough for circumfixes, because a circumfix is one affix in morphology. We also need CIRCUMFIX option for correct morphological analysis.

```
# circumfixes: ~ obligate prefix/suffix combinations
# superlative in Hungarian: leg- (prefix) AND -bb (suffix)
# nagy, nagyobb, legnagyobb, legeslegnagyobb
# (great, greater, greatest, most greatest)

CIRCUMFIX X

PFX A Y 1
PFX A 0 leg/X .

PFX B Y 1
PFX B 0 legesleg/X .

SFX C Y 3
SFX C 0 obb . +COMPARATIVE
SFX C 0 obb/AX . +SUPERLATIVE
SFX C 0 obb/BX . +SUPERSUPERLATIVE
```

Dictionary:

```
1
nagy/C   [MN]
```

Analysis:

```
> nagy
nagy[MN]
> nagyobb
nagy[MN]+COMPARATIVE
> legnagyobb
nagy[MN]+SUPERLATIVE
> legeslegnagyobb
nagy[MN]+SUPERSUPERLATIVE
```

Compounds

Allowing free compounding yields decrease in precision of recognition, not to mention stemming and morphological analysis. Although lexical switches are introduced to license compounding of bases by **Ispell**, this proves not to be restrictive enough. For example:

```
# affix file
COMPOUNDFLAG X
```

```
2
foo/X
bar/X
```

With this resource, *foobar* and *barfoo* also are accepted words.

This has been improved upon with the introduction of direction-sensitive compounding, i.e., lexical features can specify separately whether a base can occur as leftmost or rightmost constituent in compounds. This, however, is still insufficient to handle the intricate patterns of compounding, not to mention idiosyncratic (and language specific) norms of hyphenation.

The **Hunspell** algorithm currently allows any affixed form of words, which are lexically marked as potential members of compounds. **Hunspell** improved this, and its recursive compound checking rules makes it possible to implement the intricate spelling conventions of Hungarian compounds. For example, using COMPOUNDWORDMAX, COMPOUNDSYLLABLE, COMPOUNDROOT, SYLLABLENUM options can be set the noteworthy Hungarian ‘6-3’ rule. Further example in Hungarian, derivate suffixes often modify compounding properties. Hunspell allows the compounding flags on the affixes, and there are two special flags (COMPOUNDPERMITFLAG and (COMPOUNDFORBIDFLAG) to permit or prohibit compounding of the derivations.

Suffixes with this flag forbid compounding of the affixed word.

We also need several Hunspell features for handling German compounding:

```
# German compounding

# set language to handle special casing of German sharp s

LANG de_DE
```

```
# compound flags

COMPOUNDBEGIN U
COMPOUNDMIDDLE V
COMPOUNDEND W

# Prefixes are allowed at the beginning of compounds,
# suffixes are allowed at the end of compounds by default:
# (prefix)?(root)+(affix)?
# Affixes with COMPOUNDPERMITFLAG may be inside of compounds.
COMPOUNDPERMITFLAG P

# for German fogemorphemes (Fuge-element)
# Hint: ONLYINCOMPOUND is not required everywhere, but the
# checking will be a little faster with it.

ONLYINCOMPOUND X

# forbid uppercase characters at compound word bounds
CHECKCOMPOUNDCASE

# for handling Fuge-elements with dashes (Arbeits-)
# dash will be a special word

COMPOUNDMIN 1
WORDCHARS -

# compound settings and fogemorpheme for ‘Arbeit’

SFX A Y 3
SFX A 0 s/UPX .
SFX A 0 s/VPDX .
SFX A 0 0/WXD .

SFX B Y 2
SFX B 0 0/UPX .
SFX B 0 0/VWXDP .

# a suffix for ‘Computer’

SFX C Y 1
SFX C 0 n/WD .

# for forbid exceptions (*Arbeitsnehmer)

FORBIDDENWORD Z

# dash prefix for compounds with dash (Arbeits-Computer)

PFX - Y 1
PFX - 0 -/P .

# decapitalizing prefix
# circumfix for positioning in compounds
```

```
PFX D Y 29
PFX D A a/PX A
PFX D Ä ä/PX Ä
.
.
PFX D Y y/PX Y
PFX D Z z/PX Z
```

Example dictionary:

```
4
Arbeit/A-
Computer/BC-
-/W
Arbeitsnehmer/Z
```

Accepted compound compound words with the previous resource:

```
Computer
Computern
Arbeit
Arbeits-
Computerarbeit
Computerarbeits-
Arbeitscomputer
Arbeitscomputern
Computerarbeitscomputer
Computerarbeitscomputern
Arbeitscomputerarbeit
Computerarbeits-Computer
Computerarbeits-Computern
```

Not accepted compoundings:

```
computer
arbeit
Arbeits
arbeits
ComputerArbeit
ComputerArbeits
Arbeitcomputer
ArbeitsComputer
Computerarbeitcomputer
ComputerArbeitcomputer
ComputerArbeitscomputer
Arbeitscomputerarbeits
Computerarbeits-computer
Arbeitsnehmer
```

This solution is still not ideal, however, and will be replaced by a pattern-based compound-checking algorithm which is closely integrated with input buffer tokenization. Patterns describing compounds come as a separate input resource that can refer to high-level properties of constituent parts (e.g. the number of

syllables, affix flags, and containment of hyphens). The patterns are matched against potential segmentations of compounds to assess wellformedness.

Unicode character encoding

Both **Ispell** and **Myspell** use 8-bit ASCII character encoding, which is a major deficiency when it comes to scalability. Although a language like Hungarian has a standard ASCII character set (ISO 8859-2), it fails to allow a full implementation of Hungarian orthographic conventions. For instance, the '--' symbol (n-dash) is missing from this character set contrary to the fact that it is not only the official symbol to delimit parenthetic clauses in the language, but it can be in compound words as a special 'big' hyphen.

MySpell has got some 8-bit encoding tables, but there are languages without standard 8-bit encoding, too. For example, a lot of African languages have non-latin or extended latin characters.

Similarly, using the original spelling of certain foreign names like *Ångström* or *Molière* is encouraged by the Hungarian spelling norm, and, since characters 'Å' and 'è' are not part of ISO 8859-2, when they combine with inflections containing characters only in ISO 8859-2 (like elative *-bl*, allative *-tl* or delative *-rl* with double acute), these result in words (like *Ångströmrl* or *Molière-tl*.) that can not be encoded using any single ASCII encoding scheme.

The problems raised in relation to 8-bit ASCII encoding have long been recognized by proponents of Unicode. It is clear that trading efficiency for encoding-independence has its advantages when it comes a truly multi-lingual application. There is implemented a memory and time efficient Unicode handling in Hunspell. In non-UTF-8 character encodings Hunspell works with the original 8-bit strings. In UTF-8 encoding, affixes and words are stored in UTF-8, during the analysis are handled in mostly UTF-8, under condition checking and suggestion are converted to UTF-16. Unicode text analysis and spell checking have a minimal (0-20%) time overhead and minimal or reasonable memory overhead depends from the language (its UTF-8 encoding and affixation).

SEE ALSO

hunspell (1), ispell (1), ispell (4)