Ocamorph

Morphological analyser
Reference Manual
Edition draft for version 0.1

Viktor Trón

IGK, Language Technology and Cognitive Systems. Universities of Edinburgh & Saarbrücken. MOKK Lab, Budapest Intitute of Technology. Budapest. v.tron@ed.ac.uk

This file documents *the Ocamorph morphological analyser* (Ocamorph). It corresponds to release 0.1 of the Ocamorph distribution.

More information about Ocamorph can be found at the MOKK Lab homepage, http://mokk.bme.hu.

Table of Contents

1 Introduction

This document presents *the Ocamorph morphological analyser* which is being developed as part of the Budapest Institute of Technology Media Education and Research Center's HunTool Natural Language Processing Toolkit. `http://mokk.bme.hu`

1.1 Ocamorph: A Short Description

Ocamorph is a morphological analyser. It is an implementation of the online layer of the Hunmorph morphological analysis architecture.

Ocamorph is language independent. It reads *ispell-type* language resources and can analyse text based on the resources. Ocamorph implements various word analysis routines such as morphological analysis and lemmatization.

> **Note:** This document is not intended to describe how to create the language resources that feed the analyer with language dependent knowledge. See Chapter 7 [Resource Specification], page 17 for a specification of the input resources. The format described there is not intended to be edited manually. Hunlex, a resource compilation tool provides a convenient description language in which you can describe morphologies and can compile the *ispell-type* resources needed by ocamorph based on various configuration options.

In particular, this document provides you with:

1. The compulsory tedium about Chapter 2 [License], page 3, Section 3.7 [Authors], page 4, Section 3.9 [Contact], page 5, Section 3.2 [Submitting a Bug Report], page 4, etc. See Chapter 3 [About], page 4.
2. The indispensable but trivial Installation notes, see Chapter 4 [Installation], page 6.
3. How to use ocamorph and Chapter 6 [Command-line Control], page 11.
4. The detailed exposition of the input language dependent resource format (see Chapter 7 [Resource Specification], page 17);
5. Information about Chapter 8 [Related Software and Resources], page 18.

1.2 Motivation

The motivation behind HunLex came from two opposing types of requirements *lexical resources* are supposed to fulfill:

1. (i) scalability, maintainability, extensibility; and
2. (ii) optimized format for the application.

The constraints in (i) favour *one central, redundancy-free, abstract, but transparent* specification, while the ones in (ii) require *possibly multiple application-specific, potentially redundant, optimized formats.*

In order to reconcile these two opposing requirements, HunLex introduces an offline layer into the word-analysis workflow, which mediates between two levels of resources:

1. a central database conforming to (i) (also primary resource, input resource),
2. various application-specific formats conforming to (ii) (also secondary or output resource)

The primary resources are supposed to reasonably designed to help human maintanance, and the secondary ones are supposed to optimize very different things ranging from file size, performance with the tool that uses it, coverage, robustness, verbosity, normative strictness depending on who uses it for what purpose.

HunLex is used to *compile* the primary resources into a particular application-specific format. This resource compilation phase is an offline process which is highly configurable so that users can fine-tune the output resources according to their needs.

By introducing this layer of offline resource compilation, maintenance, extendability, portability of lexical resources is possible without compromising your performance on specific word-analysis tasks.

Providing the environment for a sensible primary resource specification framework and managing the offline precompilation process are the *raison d'être* behind Hunlex.

2 License

Ocamorph is free software.

It is licensed under the **Creative Commons Attribution License**. To view a copy of this license, visit `http://creativecommons.org/licenses/by/2.5/` or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

There are *no restrictions on downloading* it other than your bandwidth and our slothful ways of making things available.

There are *no restrictions on use* either other than its deficiencies, clumsy features and bugs. However, this can be amended, because there are *no restrictions on modifying* it either. See also Section 3.5 [Contribution], page 4.

Freedom of use implies that any product (analysed text) that you created using ocamorph is yours and you hold the right to distribute it in any way. Consider letting us know that you used ocamorph, though; see Section 3.9 [Contact], page 5.

What is more, there are *no restrictions on redistributing* this software or modified versions of it.

3 Authors, Contact, Bugs

3.1 License? What license?

See Chapter 2 [License], page 3.

3.2 Submitting a Bug Report

If you find a bug or an undesireable feature or anything that is worth a couple of lines ranting at the authors, please go ahead and send a bugreport on the MOKK Lab bugzilla page at `http://mokk.bme.hu` or send a mail to me (see Section 3.9 [Contact], page 5).

3.3 Requesting a New Feature

So you are using hunlex and find yourself realizing that you would need a certain feature desparately but it happens not to be implemented. Go ahead and request it from the authors (see Section 3.9 [Contact], page 5) or sit silently and hope!

3.4 Praises

If you have an opinion about ocamorph and would like the authors to hear about it. See Section 3.9 [Contact], page 5.

3.5 Contribution

Ocamorph is open source development, so developpers are welcome to contribute to make it better in any imaginable way. Contact us (see Section 3.9 [Contact], page 5) to work out the details of how and what you would want to contribute to Ocamorph.

3.6 Reference

For the context of the whole huntools kit, use

```
@InProceedings{tron:etal:05,
  author =       {Viktor Tr\'on and Gy\''orgy Gyepesi and P\'eter Hal\'acsy and Andr\'as Ko
  title =        {Hunmorph: open source word analysis},
  booktitle =    {Proceedings of the ACL 2005 Workshop on Software},
  year =         2005
}
```

These and other papers can be downloaded from the MOKK Lab publications page at `http://mokk.bme.hu`

3.7 Authors

The author of ocamorph and this document is *Viktor Trón*. He can be mailed to on `v.tron@ed.ac.uk`

Hopefully more can be found on MOKK Lab's pages at `http://mokk.bme.hu`.

Viktor Trón and Péter Halácsy wrote the C binding for ocamorph.

Péter Halácsy is the author of the MacOS package of the ocamorph executable.

Attila Balogh made the MS Windows executable.

3.8 Acknowledgements

Thanks to Péter Halácsy, Dániel Varga and András Kornai who contributed various ideas and design suggestions during the development of ocamorph.

Thanks to Chrisoph Filliatre, whose ocaml modules implementing bitvectors and tries are the basis for the respective ocamorph components.

Special thanks to Dániel Varga and Péter Halácsy for extensive testing of ocamorh.

3.9 Contact

We can get in contact if you

1. Mail to *Viktor Trón* on
   `v.tron@ed.ac.uk`
2. Join the forums on `http://mokk.bme.hu`
3. Submit a bug report (see Section 3.2 [Submitting a Bug Report], page 4) or feature request (see Section 3.3 [Requesting a New Feature], page 4).

4 Installation

So you want to install ocamorph (see Chapter 1 [Introduction], page 1) from the source distribution. This document describes what and how you can install with this distribution.

4.1 Download

The latest version of the ocamorph source distribution is always available from the MOKK LAB website at `http://mokk.bme.hu/tools/ocamorph` or, if all else fails, by mailing to Viktor Trón `v.tron@ed.ac.uk`.

4.2 Supported Platforms

Ocamorph can be compiled on any platform for which there is an `ocaml` compiler (see Section 4.3 [Prerequisites], page 6). This includes all Linuxes, unices, MacOSX, MS Windows. There are binary packages for these platforms on the MOKK LAB website at `http://mokk.bme.hu/tools/ocamorph`.

4.3 Prerequisites

If you install ocamorph from the source package you need to have some other software installed on your system.

`ocaml` [Prerequisite]

Ocamorph is written in the *ocaml* programming language `http://www.ocaml.org/`. OCaml compilers are available for virtually all platforms in various package formats for free from `http://caml.inria.fr/ocaml/distrib.html`.

You will need `ocaml` version >=3.08.2 to compile ocamorph.

`make` [Prerequisite]

Installation is based on '`OCamlMakefile`' (i.e., `ocaml-make`) courtesy of Markus Mottl `http://www.ai.univie.ac.at/~markus/home/ocaml_sources.html#OCamlMakefile`

'`OCamlMakefile`' presupposes '`GNU make`' >= 3.80 but ocamorph installation works with earlier versions of GNU make as well. The installation tries to determine the make version automatically and uses a workaround Makefile in case you use make <3.80. In case this fails, set the '`Makefile`' variable `OCAMLMAKEFILE` to either '`OCamlMakefile`' (for make >=3.80) or '`make_pre3.80.OCamlMakefile`' (for make <3.80).

`C library` [Prerequisite]

If you want to compile and install the C library for ocamorph, you need '`ar`' and `ranlib`.

`Documentation` [Prerequisite]

If you want to compile the documentation for ocamorph, you need the GNU `texinfo` software documentation system installed. `ftp://ftp.gnu.org/gnu/texinfo/`

4.4 Install

ocamorph is installed in the good old way by typing

```
$ make && sudo make install
```

in the toplevel directory of the unpacked distribution. If this works, great! Go ahead to Chapter 5 [Bootstrapping], page 10.

4.5 Advanced Install

The ocamorph distribution is available in a source tarball called ‘`ocamorph.tgz`’. First you have to unpack it by typing

```
$ tar xzvf ocamorph.tgz
```

Then, you enter the toplevel directory of the unpacked distribution with

```
$ cd ocamorph
```

To compile the tools and libraries and documentation, simply type

```
$ make
```

in the toplevel directory of the distribution.

To install it (on what gets installed, see Section 4.7 [Installed Files and Directories], page 8), type

```
$ make install
```

Well, by default this would want to install things under ‘`/usr/local`’, so you have to have admin permissions. If you are not root but you are in the sudoers file with the appropriate rights, you type:

```
$ sudo make install
```

You can change the location of the installation by changing the install prefix path with

```
$ sudo make INSTALLPREFIX=‘/my/favourite/path’ install
```

Changing the location of installation for individual install targets individually is not recommended but easy-peasy if you have a clue about ‘`make`’ and ‘`Makefile`’-s. To do this you have to change the relevant ‘`Makefile`’-s in the subdirectories of the distribution. See Section 4.7 [Installed Files and Directories], page 8.

If you have problems, doubleckeck that you have the prerequisites (see Section 4.3 [Prerequisites], page 6). If you think you followed the instructions but still have problems, submit a bug report (see Section 3.2 [Submitting a Bug Report], page 4).

If you are upgrading an earlier version of ocamorph, you may want to *uninstall* the earlier one first (see Section 4.6 [Uninstall and Reinstall], page 8).

If you do not want to compile and install the ocamorph C library, then use

```
$ make CLIB=
$ make CLIB= install
```

you can also compile any of the ocaml subprojects separately by setting the `SUBPROJS` variable:

```
$ make SUBPROJS="ocamorphlib" CLIB=
```

The subprojects are called: ‘`ocamorph_debug ocamorphlib ocamorph_noassert`’ If you compile subprojects individually than you have to cleanup after the projects, cause intermediate targets are incompatible.

```
$ make SUBPROJS="ocamorphlib" cleanup
```

4.6 Uninstall and Reinstall

The install prefix is remembered in the source distribution in the file '`install_prefix`'. So after you `cd` into the toplevel directory of the distribution, you can uninstall ocamorph by typing

```
$ make uninstall
```

You can reinstall ocamorph with

```
$ make reinstall
```

at any time if you make modifications to the code or compile options.

After installation you can delete the whole source and build tree, however, in this case the install location will not be remembered so uninstall and reinstall are not available.

> **Warning:** Note that if you fiddle with changing the location of individual install targets, uninstall and resinstall will not work correctly.

4.7 Installed Files and Directories

The following files and directories are installed, paths are relative to the *install prefix* (see Section 4.4 [Install], page 7):

- '`bin/ocamorph_debug`'
  is the executable which can be run on the command line (see Chapter 6 [Command-line Control], page 11) and can be used for debugging.
- '`bin/ocamorph_debug`'
  is the executable which can be run on the command line (see Chapter 6 [Command-line Control], page 11) and cannot be used for debugging. Otherwise it has the same functionality as '`ocamorph_debug`'.
- '`bin/ocamorph`'
  the symbolic link to '`ocamorph_noassert`' and can be run on the command line (see Chapter 6 [Command-line Control], page 11).
- '`lib/ocamorph`'
  is the directrory in which the ocamorph library components are installed.
- '`lib/ocamorph/ocamorph.cmxa`'
  is the ocamorph ocaml native code library.
- '`lib/ocamorph/ocamorph.cmi`'
  is the ocamorph ocaml native code library.
- '`lib/ocamorph/libocamorph.a`'
  is the ocamorph static C library. See
- '`lib/ocamorph/ocamorph.h`'
  is the C header file for ocamorph C library.
- '`share/doc/ocamorph/`'
  is a directory containing the ocamorph documentation. Subdirectories indicate the various document formats (see Section 4.8 [Documentation], page 9) most probably including a replica of this document.

- ‘`ocamorph.1`’

  is the ocamorph man page, that describes the command-line use of ocamorph (see see Chapter 6 [Command-line Control], page 11).

4.8 Documentation

Ocamorph documentation is available in various formats. You can compile and install them from the source distribution by changing to the doc directory and typing `make`.

Available formats are

- info texinfo pages
- plaintext
- html big bundle of html
- bightml one big file
- dvi
- ps
- pdf compiling a pdf is done with `texi2pdf` (which is a prerequisite for pdf generation)

5 Bootstrapping

6 Command-line Control

This chapter describes how to use ocamorph on the command line.

6.1 Invoking ocamorph

ocamorph can be invoked by typing `ocamorph` on the command line (in a shell, terminal). On windows platforms the name of the executable is `ocamorph.exe`.

If you install ocamorph from the source distribution, then two executables are created (see see Chapter 4 [Installation], page 6):

- `ocamorph_debug`, and
- `ocamorph_noassert`

`ocamorph_debug` can be used for debugging by setting the command-line option '`debug_level`'. `ocamorph` is a symbolic link to `ocamorph_noassert`, which cannot be used for debugging. `ocamorph_noassert` is faster than `ocamorph_debug`, but apart from debugging, the two executables are functionally equivalent. The usage of ocamorph is the same with both of these executables.

6.2 Options

All command line options are preceded by '–'. Only resource options are mandatory, the rest have default values. Some options take arguments which are described in detail below. Options can be given in any order.

6.2.1 Resource Options

Ocamorph reads ispell-type resources, i.e., an affix and a dictionary file. The specification of these language resources are described in Chapter 7 [Resource Specification], page 17.

`--aff` *affix file* [Option]

affix file is the path to the affix file. This option should be used together with the `--dic` option.

`--dic` *dictionary file* [Option]

affix file is the path to the dictionary file. This option should be used together with the `--aff` option.

Typical usage of ocamorph with the `-aff` and `--dic` options is

```
$ ocamorph --aff ./morphdb_en.aff --dic ./morphdb_en.dic
```

`--bin` *binary file* [Option]

where *binary-file* is the ocamorph native format resource. If used together with the `-aff` and `--dic` options, ocamorph compiles the native format resource and writes it to *binary-file*.

If the `--bin` option is given without the `-aff` and `--dic` options, then *binary-file* is read.

The ocamorph native resource format is a dump of the internal data structures that ocamorph uses for the analysis. This means that there is no initialization overhead if ocamorph is used with the native resource.

The native resource format is actually a memory dump of the internal data structure performed by the `ocaml Marshal` module. The native format therefore is portable between platforms but not necessarily portable between versions of ocamorph compiled with a different version of the ocaml compiler (since the implementation of the Marshal module is not guaranteed to be stable). If there is a problem with reading a native format resource that was created with an incompatible version of ocamorph, the program exits with an error. In such a case, you have to make sure you either have the appropriate version of the resource or you have the fully portable text format aff and dic files and create the native format yourself with your version ocamorph. Typing

```
     $ echo | ocamorph --aff ./morphdb_en.aff --dic ./morphdb_en.dic --bin ./morphdb_en.bin
```

reads in the aff and dic files and creates the native format writing it to `./morphdb_en.bin`. Once the compatible native format resource is available, ocamorph should be started with it. This makes ocamorph start up immediately.

A typical interactive use of ocamorph looks like this:

```
     $ ocamorph --bin ./morphdb_hu.bin
     szeretlek
     > szeretlek
     szeret/VERB<PERS<OBJ<2>>>
```

`--minimize` [Option]

If this option is given the native format resource ocamorph generates is minimized.

This option should be used together with the `-aff`, `--dic` and `--bin` options (when native format resources are generated from the aff and dic files).

Ocamorph uses a coupled trie (a tree-like data-structure where the branches are labelled with characters) to store the lexicon and affix rules. If the `--minimize` option is given this trie is minimized by collapsing identical subtries basically resulting in a finite automaton.

This minimization is computationally intensive and therefore may take a very long time to perform (Minimizing the Hungarian morphological database on my MacOS X 10.4 (1.67GHz, 2Gb RAM) takes 15 minutes). Although minimized tries give slightly better runtime perfomance, their major virtue is that the resulting resource takes less space as well as less memory space when loaded.

The exact performance effects of minimization is not fully clear to me yet.

6.2.2 Algorithmic options

These options influence the behaviour of the analysis algorithm.

`--saf` [Option]

If the `--saf` (*s*top *a*t *f*irst analysis) option is given, the algorithm does not enumerate all the alternative analyses of a token, but stops after the first one is found.

Typically, this option is used when ocamorph is used for stemming in document indexing or if postprocessing of alternatives would be too expensive.

`--compounds` [Option]

If the `--compounds` option is given the algorithm also gives back compound analyses.

Only compounds that are licenced in the resource file are possible. This option serves to enable the compounds that are ligitimated in the lexical resources.

`--blocking` [Option]

If the `--blocking` option is given the algorithm gives back less analyses. A lexical (non-affixed) partial analysis always blocks one that involves affixation. Out of two partial analyses, only ones that are not equivalent are kept.

Blocking is typically used if language resources are redundant in that they contain entries which are also productively analized by affixation or compounding but the two are considered equivalent. Blocking effectively implements the idea that productive generation of an item by affixation or compounding is a fallback option in case the item is not found lexicalized.

Blocking is done directly in the algorithm (as opposed to post-processing), therefore gives better runtime performance than full analysis without blocking.

The `blocking` and `compounds` options can be used alongside in which case blocking also suppresses a compound analysis if the compound is entered as a lexical item.

6.2.3 Input and output options

`--in` *`input_file`* [Option]

the tokens to be analysed are read from *input_file*. By default input is read from standard input.

The default file format has one token per line. Alternatively one can use the `--field` option.

`--field` *`field_num`* [Option]

the input is assumed to contain lines with tabulator delimited fields. Only the string in column *field_num* is considered a token to be analysed. Columns start from 1. Somewhat in the spirit of awk, 0 means the whole line which is the default behaviour.

A common use of the `--field` option is when we want to enrich a file with record fields with morphological analysis. A typical such situation is a frequency dictionary.

`--out` *`output_file`* [Option]

the output is written to *output_file*. By default ocamorph uses the standard output.

The format of the output is
tag_preamble + *input_line* + *no_of_analyses* + *analyses*

where *analyses* is the sequence of analyses delimited by *tag_sep*.

tag_preamble and *tag_sep* can be changed via options:

`--tab_preamble` *`tag_preamble`* [Option]

specifies the tag preamble. By default the preamble string is `"> "`.

`--tab_sep` *`tag_preamble`* [Option]
: specifies the tag separator. By default the delimiter string is a newline.

Additionally we can output the number of analyses:

`--count_analyses` [Option]
: If the `--count_analyses` option is given the number of analyses is prepended to the array of analyses (delimited by *tag_sep*). By default the number of analyses is not output. Note that using this with the `--saf` (stop at first analysis) option does not make much sense.

A typical use of these input/output options is illustrated with the following command line:

```
$ cat text|tr -s ' ' '\n' | sort | uniq -c | sed 's/^ *//' | tr ' ' '\t' |\
 ocamorph --bin ./morphdb_en.bin --field 2 --tag_sep ' ' --tag_preamble '' \
--count_analyses > text.analysed.freq
```

6.2.4 Generic options

`--debug_level` *`debug_level`* [Option]
: *debug_level* is an integer that specifies the debug level, virtually the verbosity of ocamorph. The higher the number the more verbose.

Ocamorph writes messages to the standard error.

Debug levels have the following effects:

- 0: the default value. Only error messages are written to standard error
- 1: Basic messages and warnings
- 2-6: more and more logs used for debugging. Only available with `ocamorph_debug`
- <0: completely silent. Even error messages are suppressed. Errors throw exceptions and you only see them the way the ocaml runtime system verbalizes them.

`--help` [Option]
: displays the list of options and quits

`--version` [Option]
: displays the version number and quits

6.3 Man page

This last section is a verbatim include of the ocamorph manpage (which is automatically generated by `help2man`). Please be suspicious if it seems inconsistent with the previous section.

```
OCAMORPH(1)                     User Commands                    OCAMORPH(1)



NAME
       ocamorph - manual page for ocamorph 0.1
```

```
SYNOPSIS
       <command> <options>

OPTIONS
       option description (default settings)

       ------------------------------------------

       --aff  input affix file (affix.aff)

       --dic  input dictionary file (dictionary.dic)

       --bin  binary format (no)

       ALGORITHMIC OPTIONS

       --minimize
              minimize  the  trie  [gives better performance, saves space, but
              takes long to build] (no)

       --saf  stop at first analysis (no)

       --compounds
              allow compounds (no)

       --blocking
              blocking by lexicalized relative stems (no)

       INPUT OPTIONS

       --in   input from file (stdin)

       --out  output to file (stdout)

       --field
              analyse only this field (0 = whole line)

       OUTPUT OPTIONS

       --count_analyses
              outputs the number of analyses (no)

       --tag_preamble
              preamble string ("> ")

       --tag_sep
              tag separator (newline)
       GENERIC OPTIONS

       --debug_level
              debug level (0)

       --help display this list of options and quits

       --version
              displays version info and quits

       -help  Display this list of options

SEE ALSO
       The full documentation for ocamorph is maintained as a Texinfo  manual.
       If  the info and ocamorph programs are properly installed at your site,
       the command

              info ocamorph

       should give you access to the complete manual.


ocamorph 0.1                     December 2005                     OCAMORPH(1)
```

7 Resource Specification

8 Related Software and Resources

8.1 Software that can use the output of Hunlex as input

8.1.1 Huntools

8.1.2 Myspell

8.1.3 Jmorph

8.1.4 Ispell

8.2 Available resources

8.2.1 The Hungarian Morphdb Project

8.2.2 The English Morphdb Project

8.3 Hunlex's relatives

8.3.1 XFST, TWOLC, LEXC

For `xfst`, `twolc`, `lexc`, see
`http://www.xrce.xerox.com/competencies/content-analysis/fst/home.en.html`
or
`http://www.stanford.edu/~laurik/fsmbook/home.html`

Concept Index

Files Index

Variables and Options Index

Control Flags Index

(Index is nonexistent)

Frequently Asked Questions