A Fast and Portable Morphological Component for Inflectional Languages

Günter Neumann

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Stuhlsatzenhausweg 3
66123 Saarbrücken, Germany
`neumann@dfki.uni-sb.de`

1 Introduction

In this technical paper we describe MORPHIX++ the morphological component developed at the language technology lab of the DFKI GmbH. MORPHIX++ is a descendant of MORPHIX a fast classification-based morphology component for German [?; ?]. MORPHIX++ improves MORPHIX in that the classification-based approach has been combined with the well-known two-level approach, originally developed by [?; ?]. Actually, the extensions concern

- the use of tries as the unique storage device for all sort of lexical information in MORPHIX++ (e.g., for lexical entries, prefix, inflectional endings); besides the usual functionality (insertion, retrieval, deletion), a number of more complex functions are available most notably a regular trie matcher, and
- online processing of compounds which is realized by means a recursive trie traversal. During traversal two-level rules are applied for recognizing possible decompositions of the word form in question. MORPHIX++ also supports robust processing of compounds, such that compounds are recognized by means of longest matching substrings found in the lexicon; e.g., the word "adfadfeimer" will return result for "eimer" assuming that "adfadf" is no legal lexical stem.

- A generic and parameterizable output interface for MORPHIX++has been implemented which returns a normalized feature vector representation of the computed morpho-syntactic information. It is possible to parameterize the set of relevant morpho-syntactic features which supports lexical tagging and feature relaxation.
- On basis of this new interface a very efficient specialized constraint solver (called MUNIFY) has been implemented which can be used in combination with parsing techniques in order to perform unification of the morpho-syntactic information.

Thus, the basic processing strategy employed by MORPHIX++ consists of trie traversal combined with the application of finite state automata.

Currently, MORPHIX++ has been used for the German and Italian language. An English version is under way. MORPHIX++ is implemented in Lisp following ansi Allegro Common Lisp. It has been tested under Solaris, Linux, Windows 98 and Windows-NT. The German version has a very broad coverage (a lexicon of more then 120.000 stem entries), and an excellent speed (5000 words/sec without compound handling, 2800 words/sec with compound processing (where for each compound all lexically possible decompositions are computed)[1]

2 The main control flow

3 Lexicon Management

The lexicon is implemented using a Trie abstract data type. The Trie implementation comes with its own package named "TRIE". The toplevel functions are:

- MAKE-TRIE (&OPTIONAL (NAME *TRIE*)):
  creates a trie; if name is specified, then the new trie is bound to name; otherwise the default name *trie* is used Example: (make-trie *my-trie*) returns a trie bound to *my-trie*
- INSERT-ENTRY (KEY INFO LEXICON &KEY (OVERWRITE NIL) (LEAF-NAME :INFO)):

[1]Measurement has been performed on a Sun 20 using an on-line lexicon of 120.000 entries.

inserts INFO under path KEY in LEXICON using the leaf-name LEAF-NAME; if OVERWRITE is T, then the complete old INFO is overwritten

- GET-ENTRY (KEY LEXICON &KEY (LEAF-NAME :INFO)): traverses LEXICON using KEY and returns info located under LEAF-NAME, if present, otherwise returns nil
- DELETE-ENTRY (KEY LEXICON &KEY (LEAF-NAME :INFO): deletes info found under path KEY in LEXICON located under LEAF-NAME (note, this function not only deletes the information found under KEY but also deletes the path, if possible, i.e., reorganizes the tree;
- ENTRY-EXISTS-P (KEY LEXICON &KEY (LEAF-NAME ':INFO)): a boolean function which informs you whether lexicon contains information reachable by path KEY located under LEAF-NAME
- my implementation uses sequence as a general date type; thus it is possible to use either strings, vectors, or lists as the data structures for the TRIE; the current default is strings
- regular Trie matcher:

4 Processing of compounds

5 Generation with MORPHIX++

6 Flexible user interface

7 A small user guide

8 Morphological features

The following table enumerates the features together with their domain:

:cat	→	:n :v :aux :modv :a :attr-a :def :indef :prep
		:relpron :perspron :refpron :posspron :whpron
		:ord :card :vpref :adv :whadv
		:coord :subord :intp :part
:mcat	→	:n-adj :det-word
:sym	→	:open-para :closed-para :!sign :questsign
		:seperator :dot :dotdot :semikolon :comma
:comp	→	:p :c :s
:comp-f	→	:pred-used
:det	→	:none :indef :def
:tense	→	:pres :past :subjunct-1 :sibjunct-2
:form	→	:fin :infin :infin-zu :psp :prp
		:prp-zu :imp
:person	→	1 2 :2a 3 :anrede
:gender	→	:nfm :m :f :nt
:number	→	:s :p
:case	→	:nom :gen :dat :akk