Morphological Lexicon Extraction from Raw Text Data

Markus Forsberg, Harald Hammarström and Aarne Ranta
{markus,harald2,aarne}@cs.chalmers.se

Department of Computing Science
Chalmers University of Technology
Sweden

Abstract. The tool *extract* enables the automatic extraction of lemma-paradigm pairs from raw text data. The tool uses search patterns that consist of regular expressions and propositional logic. These search patterns define sufficient conditions for including lemma-paradigm pairs in the lexicon, on the basis of word forms occurring in the data. This paper explains the search pattern syntax of *extract* as well as the search algorithm, and discusses the design of search patterns from the recall and precision point of view.
The *extract* tool was developed for morphologies defined in the *Functional Morphology* tool [1], but it is usable for all systems that implement a word-and-paradigm description of a morphology.
The usefulness of the tool is demonstrated by a case study on the Canadian Hansards Corpus of French. The result is evaluated in terms of precision of the extracted lemmas and statistics on coverage and rule productiveness. Competitive extraction figures show that human-written rules in a tailored tool is a time-efficient approach to the task at hand.

1 Introduction

A wide-coverage morphological lexicon is a key part of any information retrieval system, machine translation engine and of a variety of other Natural Language Processing applications. The demand is high not only for low-density languages, since existing lexica for major languages are often not publicly available. Moreover, even if they were, running text – especially newspapers and technical texts – will always contain new, not necessarily hapax, words.

Manual development of a full-scale lexicon is a time-consuming task, so it is natural to investigate how the lexicon development can be automated. The situation is usually such that access to large collections of raw language data is cheap, so cheap that it is tempting to look at ways to exploit the raw data to obtain the sought after high-quality morphological lexicon. Clearly, attempts to fully automatize the process (e.g [2, 3] – most other systems for unsupervised learning of morphology cannot be used directly to build a lexicon) do not reach the kind of quality we are generally interested in. However, instead of using humans for supervised learning of lexicon extraction in some form, we believe

there is a more advantageous placement of the human role. With a suitable tool, humans can use their knowledge to guide a computerized extraction from raw text, with comparatively little time spent.

To be more specific, we intend to show that a profitable role for the human is to write intelligent extraction rules. The *extract* tool has been developed with this in mind. The idea behind *extract* is simple: start with a large-sized corpus and a description of the word forms in the paradigms with the varying parts, which we refer to as *technical stems*, represented with variables. In the tool's syntax, we could describe the first declension noun of Swedish with the following definition.

```
paradigm decl1 =
  x+"a"
  { x+"a"  & x+"as"  & x+"an"  & x+"ans" &
    x+"or" & x+"ors" & x+"orna & x+"ornas" } ;
```

Given that all forms in the curly brackets, called the *constraint*, are found for some prefix `x`, the tool outputs the *head* `x+"a"` tagged with the name of the paradigm. E.g., if these forms exist in the text data: `ärta`, `ärtas`, `ärtan`, `ärtans`, `ärtor`, `ärtors`, `ärtorna` and `ärtornas`, the tool will output `decl1 ärta`. Given that we have the lemma and the paradigm class label, it is a relatively simple task to generate all word forms.

The paradigm definition has a major drawback: very few lemmas appear in all word forms. It could in fact be relaxed to increase recall without sacrificing precision: to identify a Swedish word as a noun of the first declension it is often enough to find one instance of the four singular forms and one of the four plural forms. The tool offers a solution by supporting propositional logic in the constraint, further described in section 2.1. Various issues of the extraction process are discussed in section 3.

Another problem with the given definition is the lack of control over what the variable `x` might be. Section 2.2 describes how the tool improves this situation by allowing variables to be associated with regular expressions.

The stems of first declension nouns in Swedish are the same for all word forms, but this is not the case for many paradigms, e.g. German nouns with umlaut. Section 2.3 presents the tool's use of multiple variables as a solution to this problem.

⟨Def⟩ ::= `paradigm` ⟨Name⟩ ⟨VarDef⟩ =
⟨Head⟩ { ⟨Logic⟩ }

| `regexp` ⟨Name⟩ = ⟨Reg⟩

Fig. 1. Regexp and paradigm definitions

2 Paradigm File Format

A paradigm file consists of two kinds of definitions: `regexp` and `paradigm`. The syntax is given in figure 1.

A `regexp` definition associates a name (`Name`) with a regular expression (`Reg`). A `paradigm` definition consists of a name (`Name`), a set of variable-regular expression associations (`VarDef`), a set of output constituents (`Head`) and a constraint (`Logic`).

The basic unit in `Head` and `Logic` is a *pattern* that describes a word form. A pattern consists of a sequence of variables and string literals glued together with the '+' operator. An example of a pattern given previously was `x+"a"`.

Both definitions will be discussed in detail in the following sections.

2.1 Propositional Logic

Propositional logic appears in the constraint to enable a more fine-grained description of what word forms the tool should look for. The basic unit is a pattern, corresponding to a word form, which is combined with the operators `&` (*and*), `|` (*or*), and `~` (*not*).

The syntax for propositional logic is given in figure 2, where *Pattern* refers to one word form.

⟨Logic⟩ ::= ⟨Logic⟩ `&` ⟨Logic⟩
| ⟨Logic⟩ `|` ⟨Logic⟩
| ⟨Logic⟩
| `~` ⟨Logic⟩
| ⟨Pattern⟩
| `(` ⟨Logic⟩ `)`

Fig. 2. Propositional logic grammar

The addition of new operators allow the paradigm in section 1 to be rewritten with disjunction to reflect that it is sufficient to find one singular and one plural word form.

```
paradigm decl1 =
      x+"a"
      { (x+"a"   | x+"as"  | x+"an"  | x+"ans") &
        (x+"or"  | x+"ors" |x+"orna  | x+"ornas") } ;
```

2.2 Regular Expressions

It was mentioned in section 1 that control over the variable part of a paradigm description was desired. The solution provided by the tool is to enable the user to associate every variable with a regular expression. The association dictates which (sub-)strings a variable can match. An unannotated variable can match any string, i.e. its regular expression is Kleene star over any symbol.

As a simple example, consider German, where nouns always start with an uppercase letter. This can be expressed as follows.

```
regexp UpperWord = upper letter*;

paradigm n [x:UpperWord] = ... ;
```

The syntax of the tool's regular expressions is given in figure 3, with the normal connectives: union, concatenation, set minus, Kleene star, Kleene plus and optionality. *eps* refers to the empty string, *digit* to $0-9$, *letter* to an alphabetic Unicode character, *lower* and *upper* to a lowercase respectively an uppercase letter. *char* refers to any character. A regular expression can also contain a double-quoted string, which is interpreted as the concatenation of the characters in the string.

⟨Reg⟩ ::= ⟨Reg⟩ | ⟨Reg⟩
| ⟨Reg⟩ − ⟨Reg⟩
| ⟨Reg⟩ ⟨Reg⟩
| ⟨Reg⟩ *
| ⟨Reg⟩ +
| ⟨Reg⟩ ?
| eps
| ⟨Char⟩
| digit
| letter
| upper
| lower
| char
| ⟨String⟩
| (⟨Reg⟩)

Fig. 3. Regular expression

2.3 Multiple Variables

Not all paradigm definitions are as neat as the initial example — phenomena like *umlaut* require an increased control over the variable part. The solution the tool provides is to allow multiple variables, i.e. a pattern may contain more than one variable. This is best explained with an example, where two German noun paradigms are described, both with umlaut. The change of the stem vowel is captured by introducing two variables and by letting the stem vowel be a constant string.

```
regexp Consonant = ... ;

regexp Pre = upper letter*;

regexp Aft = Consonant+ ;

paradigm n2 [F:Pre, ll:Aft] =
  F+"a"+ll
    { F+"a"+ll & F+"ä"+ll+"e" } ;

paradigm n3 [W:Pre, rt:Aft] =
   W+"o"+rt
    { W+"o"+rt & W+"ö"+rt+"er" } ;
```

The use of variables may reduce the time-performance of the tool, since every possible variable binding is considered. The use of multiple variables should be moderate, and the variables should be restricted as much as possible by their regular expression association to reduce the search space.

A variable does not need to occur in every pattern, but the tool only performs an initial match with patterns containing all variables. The reason for this is efficiency — the tool only considers one word at the time, and if the word matches one of the patterns, it searches for all other patterns with the variables instantiated by the initial match. For obvious reasons, an initial match is never performed under a negation, since this would imply that the tool searches for something it does not want to find.

It is allowed to have repeated variables, i.e. non-linear patterns, which is equivalent to *back reference* in the programming language Perl. An example where a sequence of bits is reduplicated is given. This language is known to be non-context-free [4].

```
regexp ABs = (0|1)*;

paradigm reduplication [x:ABs] =
   x+x { x+x } ;
```

2.4 Multiple Arguments

The head of a paradigm definition may have multiple arguments to support more abstract paradigms. An example is Swedish nouns, where many nouns can be correctly classified by just detecting the word forms in nominative singular and nominative plural. An example is given below, where the first and second declension is handled with the same paradigm function, where the head consists of two output forms. The constraints are omitted.

```
paradigm regNoun =
flick+"a" flick+"or"
{...} ;
```

```
paradigm regNoun =
pojk+"e" pojk+"ar"
{...} ;
```

2.5 The Algorithm

The underlying algorithm of the tool is presented in pseudo-code notation.

```
let L be the empty lexicon.
let P be the set of extraction paradigms.
let W be all word types in the corpus.
for each w : W
 for each p : P
  for each constraint C with which w matches p
   if W satisfies C with the result H,
    add H to L
   endif
  end
 end
end
```

The algorithm is initialized by reading the word types of the corpus into an array W. A word w *matches* a paradigm p, if it can match any of the patterns in the paradigm's constraint that contains all variables occurring in the constraint. The result of a successful match is an *instantiated constraint* C, i.e. a logical formula with words as atomic propositions. The corpus W *satisfies* a constraint C if the formula is true, where the truth of an atomic proposition a means that the word a occurs in W.

2.6 The Performance of the Tool

The extraction tool is implemented in Haskell. It is available as an open-source free software [1]. A typical example of using the tool, the experiment reported in Section 4 extracted a lexicon of 19,295 lemmas from a corpus of 66,853 word

[1] Extract homepage: `http://www.cs.chalmers.se/~markus/extract/`

types, by using 43 paradigms. The execution time was 11min 23s on a computer with an AMD 3600+ CPU and 1 GB memory, running Kubuntu Linux 5.10. The memory consumption was 34 MB.

3 The Art of Extraction

The constraint of a paradigm describes a sub-paradigm, a subset of the word forms, considered to be evidence enough to be able to judge that the lemmas in the head are in that paradigm class. The identification of appropriate sub-paradigms requires good insights into the target language and intuitions about the distributions of the word forms. However, these insights and intuitions may be acquired while using the tool by trial and error.

Lexicon extraction is a balance between *precision*, i.e. the percentage of the extracted lemmas that are correctly classified, and *recall*, i.e. the percentage of the lemmas in the text data that are extracted. Precision, however, is by far the most important, since poor recall can be compensated with more text data, but poor precision requires more human labor.

How about extracting the paradigm descriptions from a set of paradigms automatically? We use the term *minimum-size sub-paradigm* to describe the minimum-sized set of word forms needed to uniquely identify a paradigm P. More formally, a minimum-sized sub-paradigm is a minimum-size set of word forms $P' \subseteq P$ such that for any other paradigm Q, $P' \not\subseteq Q$. It turns out that the problem of finding the minimum-size sub-paradigm for a paradigm P is NP-complete[2]. Furthermore, the minimum-size sub-paradigm need not be of high practical interest since it may contain forms that are very uncommon in actual usage. Therefore there is all the more reason to let a human choose which forms to require and also weigh in which forms are likely to be common or uncommon in actual usage.

Also, some natural languages have *overshadowed paradigms*, i.e. paradigms where the form of one paradigm is a subset of another paradigm. For example, in Latin some noun paradigms are overshadowed by adjective paradigms. The distinction of Latin nouns and adjectives can be done through the use of negation where a second declension noun paradigm is defined by also stating that the feminine endings, which would indicate that it is an adjective, should not be present. This definition, however, misses e.g *filius* where the feminine parallel *filia* does exist.

```
paradigm decl2fungus =
```

[2] The minimum-size sub-paradigm problem (MSS) is equivalent to the well-known set-cover problem. Proof omitted.

```
    fung++"us"
{ fung+"us" & fung+"i" & ~(fung+"a" | fung+"ae")};
```

Negation is similar with *negation as failure* in Prolog, with the same problems associated with it. The main problem is that negation rests on the absence, not the presence, of information, which in turn means that the extraction process with negation is non-monotonic: the use of a larger corpus may lead to an extracted lexicon which is smaller. A worst-case scenario is a misspelt or foreign word that, by negation, removes large parts of the correctly classified lemmas in the extracted lexicon.

In most cases, a better alternative to negation is a more careful use of regular expressions, and in cases like Latin nouns, a rudimentary POS tagger that resolves the POS ambiguity may outperform negation.

3.1 Manual Verification

Almost all corpora have misspellings which may lead to false conclusions. Added to that are word forms that incidentally coincide. One possible solution to handle misspellings is to only consider words that occur at some frequency. However, that would remove a lot of unusual but correctly spelled words (to an extent which is unacceptable). Coincidences are in practice impossible to avoid.

Misspellings, foreign words and coincidences are the reason why manual verification of the extracted lexicon cannot be circumvented even with ”perfect” paradigm definitions. However, browse-filtering a high-precision extracted lexicon requires much less time than building the same lexicon by hand. Also, nothing in principle prohibits statistical techniques to be applied in collaboration here. For instance, one can sort the extracted lemmas heuristically according to how many forms and with what frequencies they occur (cf. section 5). In general, this is productive for poly-occurring lemmas but helps little for the (typically many) hapax lemmas.

4 Experiments

We will evaluate our proposed extraction technique with a study of real-world extraction on the Hansards corpus of Canadian French [5]. All words were manually annotated to enable a thorough evaluation. However, the intended practical usage of the extraction tool is to simply run the tool on the raw text data and eye-browse the output list for erroneous extractions.

The corpus consisted of approximately 15 million running tokens of 66853 types. From these 66853 types we manually removed all junk – foreign words,

proper names, misspellings, numeric expressions, abbreviations as well as pronouns, prepositions, interjections and non-derived adverbs – so that a 49477 true lexical items remained. 27681 lemmas account for the 49477 forms, where verb lemmas tended to occur in more forms than noun and adjective lemmas. Of course, not all these lemmas occurred in such forms that their morphological class could be recognized by their endings alone. Many lemmas occur in only one form – usually not enough to infer its morphological class – unless, as is often the case, they contain a derivational morpheme which, together with its inflectional ending, does suffice. For example, a single occurrence of a word ending in *-e* is hardly conclusive, whereas one ending in *-tude* is almost certainly a feminine noun with a plural in *-s*. Nouns without derivational ending cannot be reliably distinguished from adjectives even when they occur in all their forms, i.e both the singular and plural. The table in figure 4 summarizes these data.

Tokens	15 000 000
Types	66 853
Non-junk types	49 477
Lemmas	27 681

Fig. 4. Statistics on the corpus of Canadian French Hansards used in the experiment

We now turn to the question of precision and coverage of rule-extraction of the targeted 27 681 lemmas. We quickly devised a set of 43 rules to extract French nouns (18 rules), verbs (7 rules) and adjectives (18 rules). The verb-rules aimed at *-ir* and *-er* verbs by requiring salient forms for these paradigms, whereas the noun- and adjective rules make heavy use of regularities in derivational morphology to overcome the problems of overlapping forms. Two typical example groups are given below:

```
regexp NOTi = char* (char-"i") ;

paradigm Ver [regard:NOTi]
  = regard+"er"
    {regard+"e" &
     (regard+"é"   | regard+"ée"  |
      regard+"ez"  | regard+"ont" |
      regard+"ons" | regard+"a" )} ;

 paradigm Aif
  = sport+"if"
    {sport+"if"  | sport+"ifs" |
     sport+"ive" | sport+"ives"} ;
```

The results of the extraction are shown in figure 5. If possible, one would like to know where one's false positives come from – sloppy rules or noisy data? At least one would like to know roughly what to expect. Since we have already annotated this corpus we can give some indicative quantitative data. To assess the impact of misspellings and foreign words – the two main sources for spurious extractions – we show the results of the same extraction performed on the corpus *with all junk removed beforehand*. As expected, false positives increase when junk is added. To be more precise, we get a lot of spurious verbs from English words and proper names in *-er* (e.g farmer, worchester) as well as many nouns, whose identification requires only one form, from misspellings (e.g qestion). Non-junk-related cases of confusion worth mentioning are nouns in *-ment* – the same ending as adverbs – and verbs which have spelling changes (manger-mangeait, appeler-appelle etc).

	Extr. All	Extr. Non-Junk
False Positives	2 031	664
Correctly Indentified	17 264	17 264
	19 295	17 928
Precision	89.5%	96.3%

Fig. 5. Extraction results on raw text vs. text with junk removed first.

The rule productiveness, i.e a rule on average catches $17264/43 \approx 401$, must be considered very high. As for coverage, we can see that our rules catch the lions share of the available lemmas, 17 264 out of 27 681 (again, not all of which occur in enough forms to predict their morphological class), in the corpus. This is relevant because even if we can always find more raw text cheaply, we want our rules to make maximal use of whatever is available and more raw data is of little help unless we can actually extract a lot of its lemmas with reasonable effort. It is also relevant because a precision figure without a rule productiveness figure is meaningless. It would be easy to tailor 43 rules to perfect precision, perhaps catching one lemma per rule, so what we show is that precision and rule productiveness can be simultaneously high. In general it is of course up to the user how much of the raw-data lemmas to sacrifice for precision and rule-writing effort, which are usually more important objectives.

5 Related Work

The most important work dealing with the very same problem as addressed here, i.e extracting a morphological lexicon given a morphological description, is the

study of the acquisition of French verbs and adjectives in Clément et al. [6]. Likewise, they start from an existing inflection engine and exploit the fact that a new lemma can be inferred with high probability if it occurs in raw text in predictable morphological form(s). Their algorithm ranks hypothetical lemmas based on the frequency of occurrence of its (hypothetical) forms as well as part-of-speech information signalled from surrounding closed-class words. They do not make use of human-written rules but reserve an unclear, yet crucial, role for the human to hand-validate parts of output and then let the algorithm re-iterate. Given the many differences, the results cannot be compared directly to ours but rather illustrate a complementary technique.

Tested on Russian and Croat, Oliver et al. [7, 8, Ch. 3] describe a lexicon extraction strategy very similar to ours. In contrast to human-made rules, they have rules extracted from an existing (part of) a morphological lexicon and use the number of inflected forms found to heuristically choose between multiple lemma-generating rules (additionally also querying the Internet for existence of forms). The resulting rules appear not at all as sharp as hand-made rules with built-in human knowledge of the paradigms involved and their respective frequency (the latter being crucial for recall). Also, in comparison, our search engine is much more powerful and allows for greater flexibility and user convenience.

For the low-density language Assamese, Sharma et al. [3] report an experiment to induce both morphology, i.e the set of paradigms, and a morphological lexicon at the same time. Their method is based on segmentation and alignment using string counts only – involving no human annotation or intervention inside the algorithm. It is difficult to assess the strength of their acquired lexicon as it is intertwined with induction of the morphology itself. We feel that inducing morphology and extracting a morphological lexicon should be performed and evaluated separately. Many other attempts to induce morphology, usually with some human tweaking, from raw corpus data (notably Goldsmith [9]), do not aim at lexicon extraction in their current form.

There is a body of work on inducing verb subcategorization information from raw or tagged text (see [10–12] and references therein). However, the parallel between subcategorization frame and morphological class is only lax. The latter is a simple mapping from word forms to a paradigm membership, whereas in verb subcategorization one also has the onus discerning which parts of a sentence are relevant to a certain verb. Moreover, it is far from clear that verb subcategorization comes in well-defined paradigms – instead the goal may be to reduce the amount of parse trees in a parser that uses the extracted subcategorization constraints.

6 Conclusions and Further Work

We have shown that building a morphological lexicon requires relatively little human work. Given a morphological description, typically an inflection engine and a description of the closed word classes, such as pronouns and prepositions, and access to raw text data, a human with knowledge of the language can use a simple but versatile tool that exploits word forms alone. It remains to be seen to what extent syntactic information, e.g part-of-speech information, can further enhance the performance. A more open question is whether the suggested approach can be generalized to collect linguistic information of other kinds than morphology, such as e.g verb subcategorization frames.

References

1. Forsberg, M., Ranta, A.: Functional Morphology. Proceedings of the Ninth ACM SIGPLAN International Conference of Functional Programming, Snowbird, Utah (2004) 213–223
2. Creutz, M., Lagus, K.: Inducing the morphological lexicon of a natural language from unannotated text. In: Proceedings of the International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning (AKRR '05), 15-17 June, Espoo, Finland, Espoo (2005) 106–113
3. Utpal Sharma, J.K., Das, R.: Unsupervised learning of morphology for building lexicon for a highly inflectional language. In: Proceedings of the 6th Workshop of the ACL Special Interest Group in Computational Phonology (SIGPHON), Philadelphia, July 2002, Association for Computational Linguistics (2002) 1–10
4. Hopcroft, J., Ullman, J.: Introduction to Automata Theory, Languages, and Computation, Second Edition. Addison-Wesley (2001)
5. Germann, U.: Corpus of hansards of the 36th parliament of canada. Provided by the Natural Language Group of the Universtity of Southern California Information Sciences Institute. Downloadable at `http://www.isi.edu/natural-language/download/hansard/`, accessed 1 Nov 2005. (2003) 15 million words.
6. Clément, L., Sagot, B., Lang, B.: Morphology based automatic acquisition of large-coverage lexica. In: Proc. of LREC'04, Lisboa, Portugal (2004) 1841–1844
7. Oliver, A., Tadić, M.: Enlarging the croatian morphological lexicon by automatic lexical acquisition from raw corpora. In: Proc. of LREC'04, Lisboa, Portugal (2004) 1259–1262
8. Oliver, A.: Adquisició d'informació lèxica i morfosintàctica a partir de corpus sense anotar: aplicació al rus i al croat. PhD thesis, Universitat de Barcelona (2004)
9. Goldsmith, J.: Unsupervised learning of the morphology of natural language. Computational Linguistics **27**(2) (2001) 153–198
10. Kermanidis, K.L., Fakotakis, N., Kokkinakis, G.: Automatic acquisition of verb subcategorization information by exploiting minimal linguistic resources. International Journal of Corpus Linguistics **9**(1) (2004) 1–28
11. Faure, D., Nédellec, C.: Asium: Learning subcategorization frames and restrictions of selection. In Kodratoff, Y., ed.: 10th Conference on Machine Learning (ECML 98) – Workshop on Text Mining, Chemnitz, Germany, Avril 1998. Springer-Verlag, Berlin (1998)
12. Gamallo, P., Agustini, A., Lopes, G.P.: Learning subcategorisation information to model a grammar with "co-restrictions". Traitement Automatique des Langues **44**(1) (2003) 93–177