Practical Bootstrapping of Morphological Analyzers

Kemal Oflazer[1,2]
[1]Department of Computer Engineering
Bilkent University
Bilkent, Ankara, 06533,Turkey
ko@crl.nmsu.edu

Sergei Nirenburg[2]
[2]Computing Research Laboratory
New Mexico State University
Las Cruces, NM, 88003
sergei@crl.nmsu.edu

Abstract

This paper presents a semi-automatic technique for developing broad-coverage finite-state morphological analyzers for any language. It consists of three components-elicitation of linguistic information from humans, a machine learning bootstrapping scheme and a testing environment. The three components are applied iteratively until a threshold of output quality is attained. The initial application of this technique is for morphology of low-density languages in the context of the Expedition project at NMSU CRL. This elicit-build-test technique compiles lexical and inflectional information elicited from a human into a finite state transducer lexicon and combines this with a sequence of morphographemic rewrite rules that is induced using transformation-based learning from the elicited examples. The resulting morphological analyzer is then tested against a test suite, and any corrections are fed back into the learning procedure that builds an improved analyzer.

Introduction

The Expedition project is devoted to fast "ramp-up" of machine translation systems from less studied, so-called "low-density" languages into English. One of the components that must be acquired and built during this process is a morphological analyzer for the source low-density language. Since we expect that the source language informant will not be well-versed in computational linguistics in general or in recent approaches to building morphological analyzers (e.g., [Koskenniemi, 1983], [Antworth, 1990], [Karttunen *et al.*, 1992], [Karttunen, 1994]) and the operation of state-of-the-art finite state tools (e.g., [Karttunen, 1993], [Karttunen and Beesley, 1992], [Karttunen *et al.*, 1996]) in particular, the generation of the morphological analyzer component has to be accomplished almost semi-automatically. The user must be guided through a knowledge elicitation procedure for the knowledge required for the morphological analyzer. This is accomplished using the elicitation component of Expedition, the Boas system. As this task is not easy, we expect that the development of the morphological analyzer will be an iterative process, whereby the human informant will revise and/or refine the information previously elicited based on the feedback from a test runs of the nascent analyzer.

The work reported in this paper describes the use of machine learning in the process of building and refining morphological analyzers. The main use of machine learning in our current approach is in the automatic learning of formal rewrite or replace rules for morphographemic changes from the examples provided by the informant. This subtask of accounting for such phenomena is perhaps one of the more complicated aspects of building an analyzer and by automating it we expect to gain a certain improvement in productivity.

There have been a number of studies on inducing morphographemic rules from a list of inflected words and a root word list. Johnson [1984] presents a scheme for inducing phonological rules from surface data, mainly in the context of studying certain aspects of language acquisition. The premise is that languages have a finite number of alternations to be handled by morphographemic rules and a fixed number of contexts in which they appear; so if there is enough data, phonological rewrite rules can be generated to account for the data. Rules are ordered by some notion of "surfaciness", and at each stage the most surfacy rule – the rule with the most transparent context is selected. Golding and Thompson[1985] describe an approach for inducing rules of English word formation from a given corpus of root forms and the corresponding inflected forms. The procedure described there generates a sequence of transformation rules,[1] each specifying how to perform a particular inflection.

More recently, Theron and Cloete [1997] have pre-

[1]Not in the sense it is used in transformation-based learning [Brill, 1995].

sented a scheme for obtaining two-level morphology rules from a set of aligned segmented and surface pairs. They use the notion of string edit sequences assuming that only insertions and deletions are applied to a root form to get the inflected form. They determine the root form associated with an inflected form (and consequently the suffixes and prefixes) by exhaustively matching against all root words. The motivation is that "real" suffixes and prefixes will appear often enough in the corpus of inflected forms, so that, once frequently occurring suffixes and prefixes are identified, one can then determine the segmentation for a given inflected word by choosing the segmentation with the most frequently occurring affix segments and considering the remainder to be the root. While this procedure seems to be reasonable for a small root word list, the potential for "noisy" or incorrect alignments is quite high when the corpus of inflected forms is large and the procedure is not given any prior knowledge of possible segmentations. As a result, selecting the "correct" segmentation automatically becomes quite nontrivial. An additional complication is that allomorphs show up as distinct affixes and their counts in segmentations are not accumulated, which might lead to actual segmentations being missed due to fragmentation. The rule induction is not via a learning scheme: aligned pairs are compressed into a special data structure and traversals over this data structure generate morphographemic rules. Theron and Cloete have experimented with pluralization in Afrikaans, and the resulting system has shown about 94% accuracy on unseen words.

Goldsmith [1998] has used an unsupervised learning method based on the minimum description length principle to learn the "morphology" of a number of languages. What is learned is a set of "root" words and affixes, and common inflectional pattern classes. The system requires just a corpus of words in a language. In the absence of any root word list to use as a scaffolding, the shortest forms that appear frequently are assumed to be roots, and observed surface forms are then either generated by concatenative affixation of suffixes or by rewrite rules.[2] Since the system has no notion of what the roots and their part of speech values really are, and what morphological information is encoded by the affixes, these need to be retrofitted manually by a human (if one is building a morphological analyzer) who would have to weed through a large number of noisy rules. We feel that this approach, while quite novel, can be used to build real-world morphological analyzers only after substantial modifications are made.

[2] Some of which may not make sense, but are necessary to account for the data: for instance a rule like *insert a word final y after the root "eas"*, is used to generate *easy*.

This paper is organized as follows: The next section very briefly describes the Boas project of which this work is a part. The subsequent sections describe the details of the approach, the morphological analyzer architecture, and the induction of morphographemic rules along with explanatory examples. Finally, we provide some conclusions and ideas for future work.

The BOAS Project

Boas [Nirenburg, 1998, Nirenburg and Raskin, 1998] is a semi-automatic knowledge elicitation system that guides a team of two people through the process of developing the static knowledge sources for a moderate-quality, broad-coverage MT system from any "low-density" language into English. Boas contains knowledge about human language and means of realization of its phenomena in a number of specific languages and is, thus, a kind of a "linguist in the box" that helps non-professional acquirers with the task. In the spirit of the goal-driven, "demand-side" approach to computational applications of language processing [Nirenburg, 1996], the process of acquiring this knowledge has been split into two steps: (i) acquiring the descriptive, declarative knowledge about a language and, (ii) deriving operational knowledge (content for the processing engines) from this descriptive knowledge. A typical elicitation interaction screen of Boas is shown in Figure 1.

An important aspect that we strive to achieve regarding these descriptive and operational pieces of information, be it elicited from human informants or acquired via machine learning is that they should be *transparent* and *human readable*, and where necessary *human maintainable and extendable,* contrary to opaque and uninterpretable representations acquired by various statistical learning paradigms.

Before proceeding any further we would also like to state the aims and limitations of our approach. Our main goal is to significantly expedite the development of a morphological analyzer. It is clear that for inflectional languages where each root word can be associated with a finite number of word forms, one can, with a lot of work, generate a list of word forms with associated morphological features encoded, and use this as a look-up table to analyze word forms in input texts. This is, however, something we would like to avoid, as it is time consuming, expensive and error-prone. We would prefer attempting to capture general morphophonological and morphographemic phenomena, and lexicon abstractions (say as inflectional paradigms) using an example driven technique, and essentially reduce the acquisition process to one of just assigning *root or citation* forms to one of these lexicon abstractions, with the automatic generation process to be described, doing the rest of

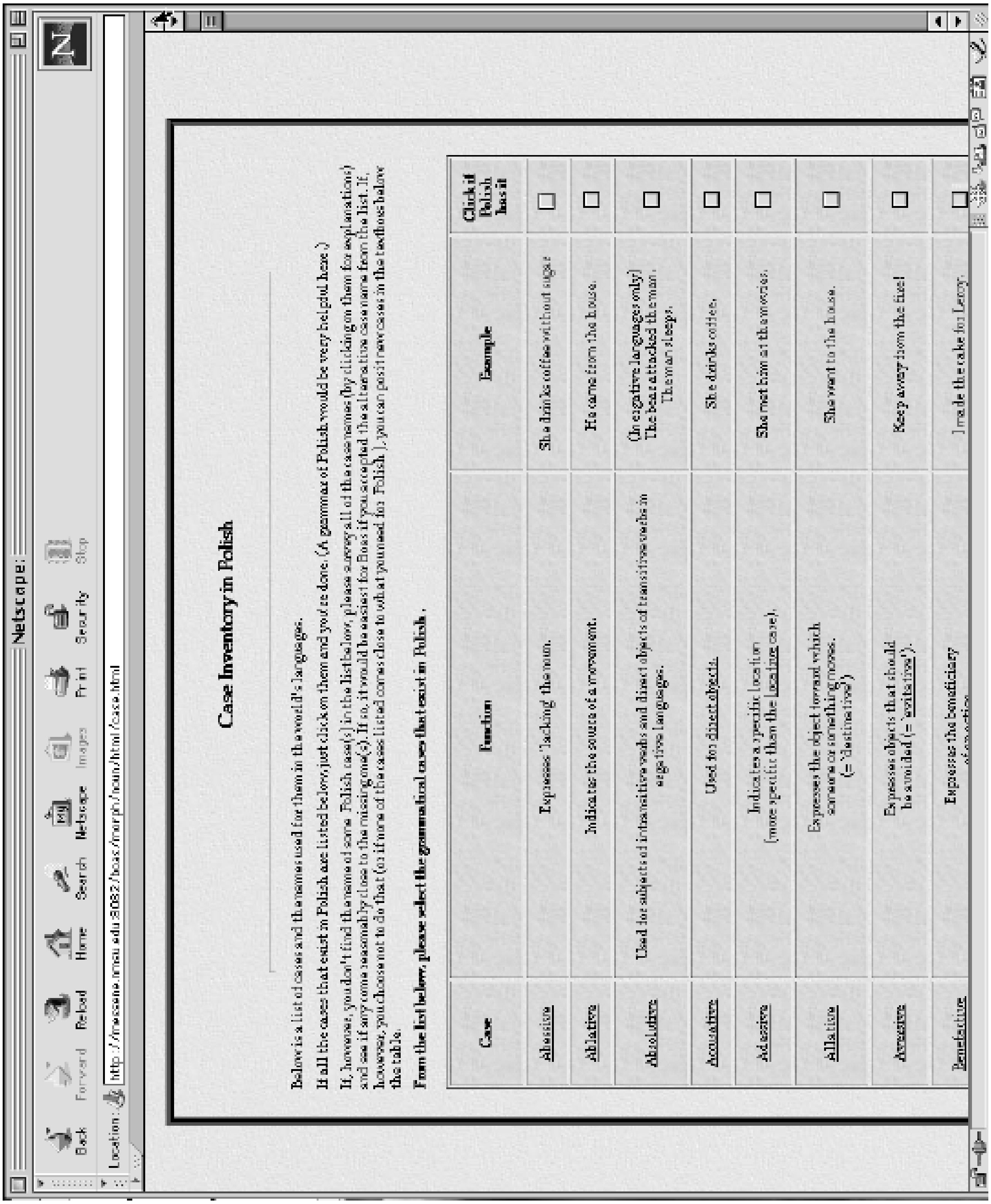

Figure 1: A sample Boas elicitation screen

the work. This process will still be imperfect, as we expect human informants to err in making their paradigm abstractions, and overlook details or exceptions. So, the whole process will be an iterative one, with convergence to a wide-coverage analyzer coming slowly at the beginning (where morphological phenomena and lexicon abstractions are being defined and tested), but significantly speeding up once wholesale root form acquisition starts. Since the generation of the operation content (data files to be used by the morphological analyzer engine) from the elicited descriptions, is expected to take a few minutes, feedback on operational performance can be provided very fast. There are also ways to utilize a partially acquired morphological analyzer to aid in the acquisition of open class root or citation forms.

Human languages have many diverse morphological phenomena and it is not our intent at this point to have a universal architecture that can accommodate any and all phenomena. Rather, we propose a modular and extensible architecture that can accommodate additional functionality in future incarnations of Boas. We also intend to limit the morphological processing to processing single tokens and deal with multi-token phenomena such as partial or full word reduplications with additional machinery that we do not discuss here.

The Elicit-Build-Test Paradigm

In this paper we concentrate on operational content in the context of building a morphological analyzer. To determine this content, we integrate the information provided by the informant with automatically derived information. The whole process is an iterative one as illustrated in Figure 2, whereby the information elicited is transformed into operational data required by the generic morphological analyzer engine[3] and the resulting analyzer is tested on a test corpus.[4] Any discrepancies between the output of the analyzer and the test corpus are then analyzed and potential sources of errors are given as feedback to the elicitation process. Currently, this feedback is limited to morphographemic processes.

The box in Figure 2 labeled *Morphological Analyzer Generation* is the main component which takes in the information elicited and generates a series of regular expressions for describing the morphological lexicon and morphographemic rules. The morphographemic rules describing changes in spelling as a result of affixation operations, are induced from the examples provided, by using transformation-based learning [Brill, 1995, Satta and Henderson, 1997]. The result is an ordered set of contextual replace or rewrite rules, much like those used in phonology. We then use error-tolerant finite state recognition [Oflazer, 1996] to perform "reverse spelling correction" for identifying the erroneous words the finite state analyzer accepts that are (very) close to the correct words in the test corpus that it rejects. The resulting pairs are then aligned, and the resulting mismatches are identified and logged for feedback purposes.

Morphological Analyzer Architecture

We adopt the general approach advocated by Karttunen [1994] and build the morphological analyzer as the combination of several finite state transducers some of which are constructed directly from the elicited information while others are constructed from the output of the machine learning stage. Since the combination of the transducers is computed at compile time, there are no run time overheads. The basic architecture of the morphological analyzer is depicted in Figure 3. The components of this generic architecture are as follows: The analyzer consists of the union of transducers each of which implements the morphological analysis process for one paradigm. Each such transducer is the composition of a number of components. These components are (from bottom to top) described below:

1. The bottom component is an ordered sequence of morphographemic rules that are learned via transformation-based learning from the examples for the inflectional paradigm provided by the human informant. The rules are then composed into one finite state transducer [Kaplan and Kay, 1994].
2. The *root and morpheme lexicon* contains the root words and the affixes. We currently assume that all affixation is concatenative and that the lexicon is described by a regular expression of the sort `[ Affixes ]* [ Roots ] [ Suffixes ]*`.[5]
3. The *morpheme to surfacy feature mapping* essentially maps morphemes to feature names but retains some encoding of the surface morpheme. Thus, allomorphs that encode the same feature would be mapped to different "surfacy" features.
4. The *lexical and surfacy constraints* specify any conditions to constrain the possibly overgenerating morphotactics of the root and morpheme lexicon. These

[3] We currently use XRCE finite state tools as our target environment [Karttunen *et al.*, 1996].

[4] Also independently elicited from either the human informant or compiled from any on-line resources for the language in question.

[5] We currently assume that we have at most one prefix and at most one suffix, but this is not a fundamental limitation. On the other hand, elicitation of complex morphotactics for an agglutinative language like Turkish or Finnish, requires a more sophisticated elicitation machinery.

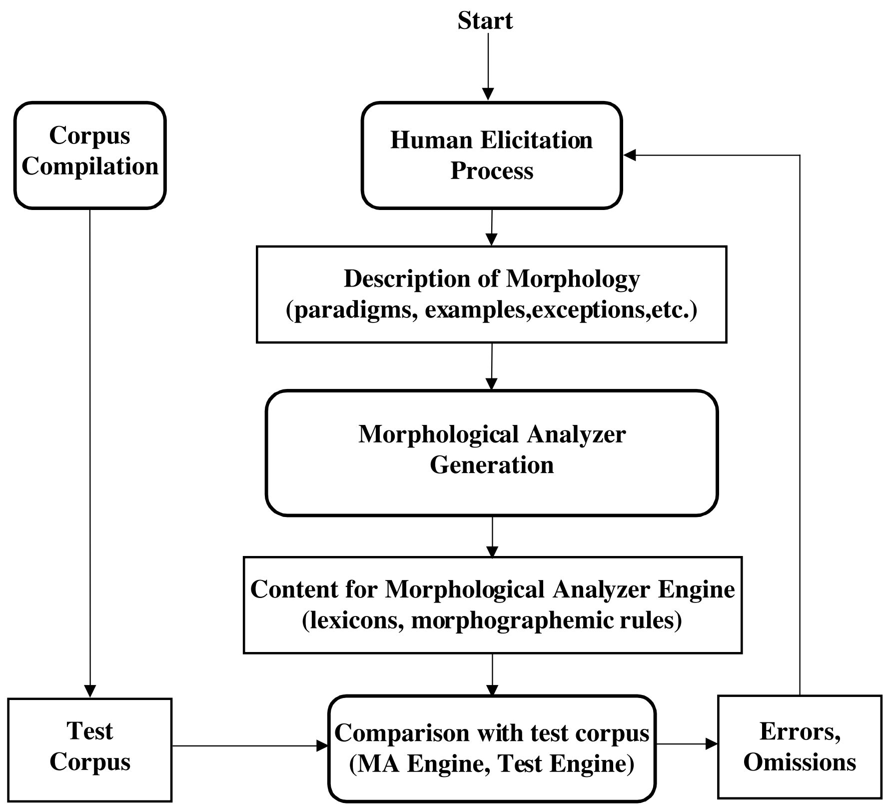

Figure 2: The Elicit-Build-Test Paradigm for Bootstrapping a Morphological Analyzer

constraints can be encoded using the root morphemes and the surfacy features generated by the previous mapping. The use of surfacy features enables reference to zero morphemes which otherwise could not have been used. For instance, if in some paradigm a certain prefix does not co-occur with a certain suffix, or always occurs with some other suffix, or if a certain root/lemma of that paradigm has exceptional behavior with respect to one or more of the affixes, or if the allomorph that goes with a certain root depends on the properties of the root, these are encoded at this level as a finite state constraint.

5. The *surfacy feature to feature mapping* module maps the surfacy representation of the affixes to symbolic feature names; as a result, no surface information remains except for the lemma or the root word. Thus, for instance, allomorphs that encode the same feature and map to different surfacy features, now map to the same feature symbol.

6. The *feature constraints* specify any constraints among the symbolic features. This is an alternative functionality to that provided by lexical and surfacy constraints to constrain morphotactics, but at this level one refers to and constrains features as opposed to surfacy features. This may provide a more natural or convenient abstraction, especially for languages with long distance morphotactic constraints.

These six finite state transducers are composed to yield the transducer for the paradigm, and the union of the resulting transducers produces one (possibly large) transducer for morphological analysis where surface strings applied at the lower side produce all possible analyses at the upper side.

Information Elicited from Human Informants

The Boas environment elicits morphological information by asking the informant a series of questions about the paradigms of inflection. A paradigm abstracts together lemmas (or root words) that essentially behave the same with respect to inflection, and captures information about the morphological features encoded and forms realizing these features, from which additional information can be extracted. It is assumed that all lemmas that belong to the same paradigm take the same set of inflectional affixes. It is expected that the roots

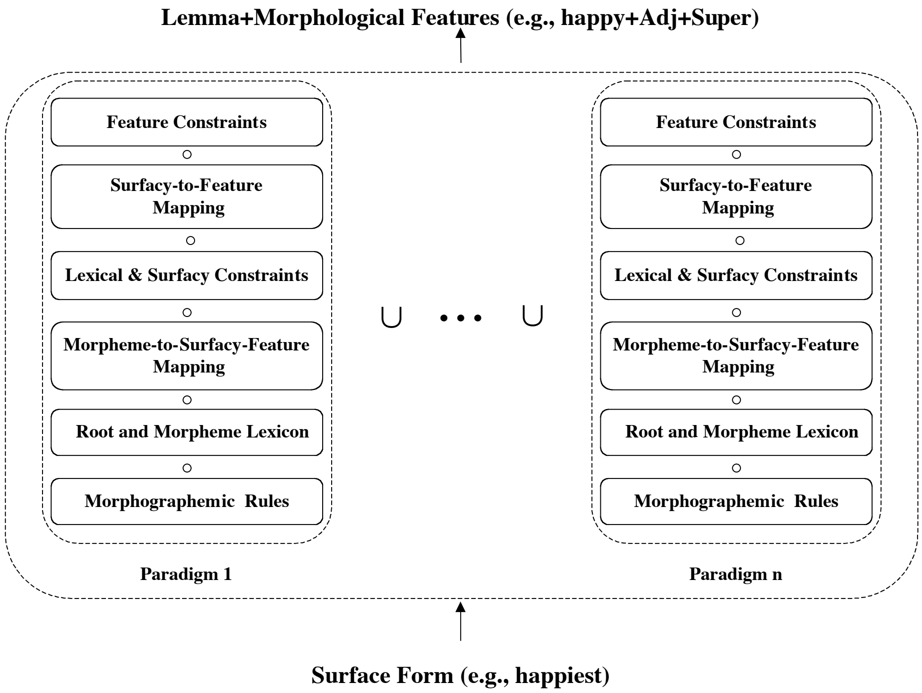

Figure 3: General Architecture of the Morphological Analyzer

and/or the affixes may undergo systematic or idiosyncratic morphographemic changes. It is also assumed that certain lemmas in a given paradigm may behave in some exceptional way (for instance, contrary to all other lemmas, a given lemma may not have one of the inflected forms!) A paradigm description also provides the full inflectional patterns for one characteristic or distinguished lemma belonging to the paradigm, and additional examples for any other lemmas whose inflectional forms undergo nonstandard morphographemic changes. If necessary, any lexical and feature constraints can be encoded. Currently the provisions we have for such constraints are limited to writing regular expressions (albeit at a much higher level), but capturing such constraints using a more natural language (e.g., [Ranta, 1998]) can be stipulated for future versions.

Preprocessing

The information elicited from the human informant is captured as a text file. The root word and the inflection examples for the distinguished lemma are processed with an alignment algorithm to determine how the given root word aligns with each inflected form so that the edit distance is minimum. Once such alignments are performed, the segments in the inflected form that are before and after the root alignment points are considered to be the prefixes and suffixes of the paradigm. These are then associated with the features given with the inflected form.

Let us provide a simple example from a Russian verb inflection paradigm. The following information about the distinguished lemma in the paradigm is provided:[6]

```
ROOT  rez     Verb        LEMMA rezat'
FORM  rezat'  Inf         FORM  reZ'    Impsg
FORM  reZ'te  Imppl       FORM  reZu    Pres1sg
FORM  reZeS   Pres2sg     FORM  reZet   Pres3sg
FORM  reZem   Pres1pl     FORM  reZete  Pres2pl
FORM  reZut   Pres3pl     FORM  rezali  PastPl
FORM  rezalo  PastNsg     FORM  rezala  PastFsg
FORM  rezal   PastMsg
```

The alignment produces the following suffix feature pairs for the suffix lexicon and morpheme to feature mapping transduction:

```
+at'-> +Inf         +' -> +Impsg      +'te -> +Imppl
+u  -> +Pres1sg     +eS -> +Pres2sg   +'et -> +Pres3sg
+em -> +Pres1pl     +ete -> +Pres2pl  +ut -> +Pres3pl
+ali -> +PastPl     +alo -> +PastNsg  +ala -> +PastFsg
+al -> +PastMsg
```

We then produce the following segmentations to be used by the learning stage discussed in the next section. It should be noted we (can) use the lemma form as the morphological stem, so that the analysis we generate will have the lemma. Thus, some of the rules learned later will need to deal with this.

[6] Upper case characters and the single quote symbol encode specific Russian characters. The transliteration is not conventional.

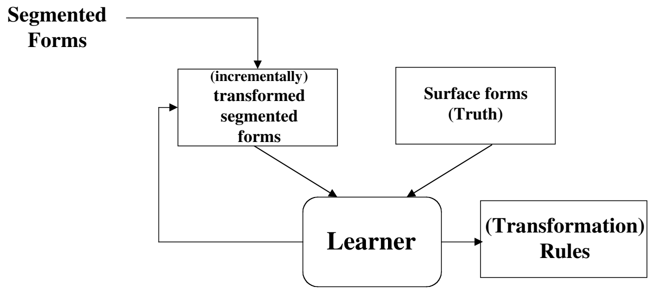

Figure 4: Transformation-based learning of morphographemic rules

```
(rezat'+at', rezat')     (rezat'+', reZ')
(rezat'+'te, reZ'te)     (rezat'+eS, reZeS)
(rezat'+et,  reZet)      (rezat'+em, reZem)
(rezat'+ete, reZete)     (rezat'+ut,  reZut)
(rezat'+ali, rezali)     (rezat'+alo, rezalo)
(rezat'+ala, rezala)     (rezat'+al, rezal)
```

Learning Segmentation and Morphographemic Rules

The lemma and suffix information elicited and extracted as summarized above are used to construct regular expressions for the lexicon component of each paradigm.[7] The example segmentations like those above are fed into the learning module to induce morphographemic rules.

Generating Candidate Rules from Examples The preprocessing stage yields a list of pairs of *segmented lexical forms*, and *surface forms*. The segmented forms have the roots/lemmas and affixes, and the affix boundaries are marked by the + symbol. This list is then processed by a transformation-based learning paradigm[Brill, 1995, Satta and Henderson, 1997] as illustrated in Figure 4. The basic idea is that we consider the list of segmented words as our input and find transformation rules (expressed as contextual rewrite rules) to incrementally transform it into the list of surface forms. The transformation we choose at every iteration is the one that makes the list of segmented forms closest to the list of surface forms.

The first step in the learning process is an initial alignment of pairs using a standard dynamic programming scheme. The only constraints in the alignment are that a + in the segmented lexical form is always aligned with an empty string on the surface side (notated by a 0), and that a consonant (vowel) on one side is aligned with a consonant (vowel) or 0 on the other side. The alignment is also constrained by the fact that it should correspond to the minimum edit distance between the original lexical and surface forms.[8] ¿From this point on, we will use a simple example from English to clarify our points.

We assume that we have the pairs (un+happy+est, unhappiest) and (shop+ed, shopped) in our example base. We align these and determine the total number of "errors" in the segmented forms that we have to fix to make all match the corresponding surface forms. The initial alignment produces the aligned pairs:

```
un+happy+est    shop0+ed
un0happi0est    shopp0ed
```

with a total of 5 errors. From each segmented pair we generate rewrite rules of the sort[9]

```
u -> l || LeftContext _ RightContext ;
```

where u(pper) is a symbol in the segmented form, l(ower) is a symbol in the surface form. Rules are generated only from those aligned symbol pairs which are different. LeftContext and RightContext are simple regular expressions describing contexts in the segmented side (up to some small length) taking into account also the word boundaries. For instance, from the first aligned-pair example, this procedure would generate rules such as (depending on the amount of left and right context allowed)

```
y -> i ||  p _               y -> i ||  p _ + e
y -> i ||  p _ + e s         y -> i ||  p _ + e s t
y -> i ||  p _ + e s t #     y -> i ||  p  p _ + e
   . . .
+ -> 0 ||  # u n  _          + -> 0 ||  # u n  _  h a p
+ -> 0 ||         _  e s t
   . . .
+ -> 0 ||  _  e s t #    . . .
+ -> 0 ||  p p y  _  e s t #
```

The # symbol denotes a word boundary, to capture any word initial and final phenomena. The segmentation rules (+ -> 0) require at least some minimal left or right context (usually longer than the minimal context for other rules for more accurate segmentation decisions). We also disallow contexts that consist only of a morpheme boundary, as such contexts are usually not informative. It should also be noted that these are rules that transform a segmented form into a surface form (contrary to what may be expected for analysis.) This lets us capture situations where multiple segmented forms may map to the same surface form,

[7] The result of this process is a script for the XRCE finite state tool *xfst*. Large scale lexicons can be more efficiently compiled by the XRCE tool *lexc*. We currently do not generate *lexc* scripts, but it is trivial to do so.

[8] We choose one if there are multiple legitimate alignments.

[9] We use the XRCE Finite State Tools regular expression syntax [Karttunen *et al.*, 1996]. For the sake of readability, we will ignore the escape symbol (%) that should precede any special characters (e.g., +) used in these rules.

which would be the case when the language has morphological ambiguity. Thus, in a reverse look-up a given surface form may be interpreted in multiple ways if applicable.[10]

Since we have many examples of aligned pairs, it is likely that a given rule will be generated from many pairs. For instance, if the pairs `(stop+ed, stopped)` and `(trip+ed, tripped)` were also in the list, the gemination rule `0 -> p || p _ + e d`, (along with certain others) will also be generated from these examples. We count how many times a rule is generated and associate this number with the rule as its *promise*, meaning that it promises to fix this many "errors" if it is selected to apply to the current list of segmented forms.

Generalizing Rules If information regarding phoneme/grapheme classes in addition to consonant and vowel classes, such as `SIBILANTS` = {s,x,z}, `LABIAL` = {b,m, ...} `HIGHWOVELS` = { u, i ...}, etc., it is possible to generate more general rules. Such rules can cover more cases and the number of rules induced will typically be smaller and cover more unseen cases. For instance, in addition to a rule like `0 -> p || p _ + e`, the rules

```
0 -> p  ||   CONSONANTS _ e
0 -> p  ||   p _ VOWELS
0 -> p  ||   LABIALS _ e
. . .
0 -> p  ||   CONSONANTS _ VOWELS
```

can be generated where symbols such as `CONSONANTS` or `LABIALS` stand for regular expressions denoting the union of relevant symbols in the alphabet. The promise scores of the generalized rules are found by adding the promise scores of the original rules generating them. It should also be noted that generalization will increase substantially the number of candidate rules to be considered during each iteration, but this is hardly a serious issue, as the number of examples one would have per paradigm would be quite small. The rules learned in the process would be the most general set of rules that do not conflict with the evidence in the examples.

Selecting Rules At each iteration all the rules along with their promise scores are generated from the current state of the example pairs. The rules generated are then ranked based on their promise scores with the top rule having the highest promise. Among rules with the same promise score, we rank more general rules higher with generality being based on context subsumption. However, all the segmentation rules go to the bottom of the list, though within this group rules are still ranked based on decreasing promise and context generality. The reasoning for treating the segmentation rules separately and later in the process, is that affixation boundaries constitute contexts for any morphographemic changes and they should not be eliminated if there are any (more) morphographemic phenomena to process.

Starting with the top ranked rule we test each rule on the segmented component of the pairs using the finite state engine, to see how much the segmented forms are "fixed". The first rule that fixes as many "errors" as it promises to fix, gets selected and is added to the list of rules generated, in order.[11]

The complete procedure for rule learning can now be given as follows:

```
- Align surface and segmented forms;
- Compute total Error;
- while(Error > 0) {
   -Generate all possible rewrite rules
    (subject to context size limits);

   -Rank Rules;

   -while (there are more rules and
           a rule has not yet been selected) {
         - Select the next rule;
         - Tentatively apply rule to
           all the segmented forms;

         - Re-align the resulting segmented
           forms with the corresponding
           surface forms to see
           how many ''errors'' have
           been fixed;

         - If the number fixed is equal to
    what the rules promised to fix
           select this rule;
      }

    -Commit the changes with the changes
     performed by the rule and
     save alignments;

    -Reduce Error by the promise
     score of the selected rule;
  }
```

This procedure eventually generates an ordered sequence of two groups of rewrite rules. The first group of rules are for any morphographemic phenomena in the given set of examples, and the second group of rules handle segmentation. All these rules are composed in the order generated to construct the *Morphographemic Rules* transducer at the bottom of each paradigm (see Figure 3).

[10] However, the learning procedure may fail to fix all errors, if among the examples there are cases where the same segmented form maps to two different surface forms (generation ambiguity).

[11] Note that a rule may actually clobber other places, since context checking is done only on the segmented form side and what it delivers may be different than what it promises, as promise scores are also dependent on the surface side.

Identifying Errors and Providing Feedback

Once the *Morphographemic Rules* transducers are compiled and composed with the lexicon transducer that is generated automatically from the elicited information, we obtain the analyzer as the union of the individual transducers for each paradigm. It is now possible to test this transducer against a test corpus and to see if there are any surface forms in the test corpus that are not recognized by the generated analyzer. Our intention is to identify and provide feedback about any minor problems that are due to a lack of examples that cover certain morphographemic phenomena, or to an error in associating a given lemma with a paradigm.

Our approach here is as follows: we use the resulting morphological analyzer with an error-tolerant finite state recognizer engine [Oflazer, 1996]. For any (correct) word in the test corpus that is not recognized we try to find words recognized by the analyzer that are (very) close to the rejected word, by error-tolerant recognition, performing essentially a reverse spelling correction. If the rejection is due a small number (1 or 2) of errors, the erroneous words recognized by the recognizer are aligned with the corresponding correct words from the test corpus. These aligned pairs can then be analyzed to see what the problems may be.

An Example

The examples generated from the above Russian paradigm will induce the following rules coded using XRCE notation and composed with `.o.` operator. ([. .] indicates empty string.):[12]

```
[t -> [. .] ||  _ ' + ] .o.
[a -> [. .] ||  _ ' + ] .o.
[z -> [. .] ||  _ ' + ] .o.
[' -> z ||  _ + a ] .o.
[' -> Z ||  _ + e ] .o.
[' -> Z ||  _ + u ] .o.
[' -> [. .] ||  _ + ' ] .o.
[. .] -> Z ||  _ + ' ] .o.
[' -> [. .] || Z +  _ e ] .o.
[+ -> [. .] ||  _ ' # ] .o.
[+ -> [. .] ||  _ u # ] .o.
[+ -> [. .] ||  _ e S # ] .o.
[+ -> [. .] ||  _ a l # ] .o.
[+ -> [. .] ||  _ e m # ] .o.
[+ -> [. .] ||  _ e t # ] .o.
[+ -> [. .] ||  _ u t # ] .o.
[+ -> [. .] ||  _ a t ' # ] .o.
[+ -> [. .] ||  _ ' t e # ] .o.
[+ -> [. .] ||  _ a l a # ] .o.
[+ -> [. .] ||  _ a l i # ] .o.
[+ -> [. .] ||  _ a l o # ] .o.
[+ -> [. .] ||  _ e t e # ]
```

Note that since we require that the analyses contain the verbal lemmas, a number of rules deal with the lemma marker `+at'`. These rules when composed with the lexicon, will, for example, output

```
rezat'+Verb Par2 +Impsg
```

in response to input `reZu`. Now, `pisat'` is a verb that was included in this paradigm, and running the corpus containing inflected forms of `pisat'` through the error-tolerant analyzer and subsequent alignment would raise the following flags (among others):

```
Morp.-> pisZut  pisZete  pisZte  piszali  piszalo
File -> piS0ut  piS0ete  piS0te  pis0ali  pis0alo
           ^^      ^^       ^^      ^        ^
```

which indicate a consistent problem due either to a wrong paradigm selection for this verb or the lack of examples that would describe the `s` → `S` alternation. Since only examples from one verb were given, some of the rules were specialized to fixing the phenomena in those examples, which explains the spurious `z/Z` in the inflected forms of `pisat'`. Adding such examples for the verb to the example base or defining a new paradigm for this other verb in the next round solves these problems.

Performance Issues The process of generating a morphological analyzer once the descriptive data is given, is very fast. Each paradigm can be processed within seconds on a fast workstation, including the few dozens of iterations of rule learning from the examples. A new version of the analyzer can be generated within minutes and tested very rapidly on any test data. Thus, none of the processes described in this paper constitutes a bottleneck in the elicitation process.

Summary and Conclusions

We have presented the highlights of our approach for automatically generating finite state morphological analyzers from information elicited from human informants. Our approach uses transformation-based learning to induce morphographemic rules from examples and combines these rules with the lexicon information elicited to compile the morphological analyzer. There are other opportunities for using machine learning in this process. For instance, one of the important issues in wholesale acquisition of open class items is that of determining which paradigm a given lemma or root word belongs to. From the examples given during the acquisition phase it is possible to induce a classifier that can perform this selection to aid the informant.

We believe that this approach to machine learning of a natural language processor that involves a human informant in an elicit-generate-test loop and uses scaffolding provided by the human informant in machine learning, is a very viable approach that avoids the noise and opaqueness of other induction schemes. Our current

[12] This example does not involve rule generalization.

work involves using similar principles to induce (light) syntactic parsers in the Boas framework.

Acknowledgements

This research was supported in part by Contract MDA904-97-C-3976 from the US Department of Defense. We also thank XRCE for providing the finite state tools.

References

[Antworth, 1990] Evan L. Antworth. *PC-KIMMO: A two-level processor for Morphological Analysis.* Summer Institute of Linguistics, Dallas, Texas, 1990.

[Brill, 1995] Eric Brill. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, 21(4):543–566, December 1995.

[Golding and Thompson, 1985] Andrew Golding and Henry S. Thompson. A morphology component for language programs. *Linguistics*, 23, 1985.

[Goldsmith, 1998] John Goldsmith. Unsupervised learning of the morphology of a natural language. Unpublished Manuscript, available at `http://humanities.uchicago.edu/ faculty/ goldsmith/index.html`, 1998.

[Johnson, 1984] Mark Johnson. A discovery procedure for certain phonological rules. In *Proceedings of 10th International Conference on Computational Linguistics–COLING'84*, 1984.

[Kaplan and Kay, 1994] Ronald M. Kaplan and Martin Kay. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378, September 1994.

[Karttunen and Beesley, 1992] Lauri Karttunen and Kenneth. R. Beesley. Two-level rule compiler. Technical Report, XEROX Palo Alto Research Center, 1992.

[Karttunen *et al.*, 1992] Lauri Karttunen, Ronald M. Kaplan, and Annie Zaenen. Two-level morphology with composition. In *Proceedings of the* 15^{th} *International Conference on Computational Linguistics*, volume 1, pages 141–148, Nantes, France, 1992. International Committee on Computational Linguistics.

[Karttunen *et al.*, 1996] Lauri Karttunen, Jean-Pierre Chanod, Gregory Grefenstette, and Anne Schiller. Regular expressions for language engineering. *Natural Language Engineering*, 2(4):305–328, 1996.

[Karttunen, 1993] Lauri Karttunen. Finite-state lexicon compiler. XEROX, Palo Alto Research Center–Technical Report, April 1993.

[Karttunen, 1994] Lauri Karttunen. Constructing lexical transducers. In *Proceedings of the* 16^{th} *International Conference on Computational Linguistics*, volume 1, pages 406–411, Kyoto, Japan, 1994. International Committee on Computational Linguistics.

[Koskenniemi, 1983] Kimmo Koskenniemi. Two-level morphology: A general computational model for word form recognition and production. Publication No: 11, Department of General Linguistics, University of Helsinki, 1983.

[Nirenburg and Raskin, 1998] Sergei Nirenburg and Victor Raskin. Universal grammar and lexis for quick ramp-up of MT systems. In *Proceedings of First International Conference on Language Resources and Evaluation*, 1998.

[Nirenburg, 1996] Sergei Nirenburg. Supply-side and demand-side lexical semantics. In *Proceedings of the Workshop on Breadth and Depth of Semantic Lexicons at the 34th Annual Meeting of the Association for Computational Linguistics*, 1996.

[Nirenburg, 1998] Sergei Nirenburg. Project Boas: "A Linguist in a Box" as a multi-purpose language resource. In *Proceedings of COLING'98*, 1998.

[Oflazer, 1996] Kemal Oflazer. Error-tolerant finite-state recognition with applications to morphological analysis and spelling correction. *Computational Linguistics*, 22(1):73–90, March 1996.

[Ranta, 1998] Aarne Ranta. A multilingual natural language interface to regular expressions. In Lauri Karttunen and Kemal Oflazer, editors, *Proceedings of International Workshop on Finite State Methods in Natural Language Processing, FSMNLP'98*, 1998.

[Satta and Henderson, 1997] Giorgio Satta and John C. Henderson. String transformation learning. In *Proceedings of ACL/EACL'97*, 1997.

[Theron and Cloete, 1997] Pieter Theron and Ian Cloete. Automatic acquisition of two-level morphological rules. In *Proceedings of 5th Conference on Applied Natural Language Processing*, 1997.