Natural Language Engineering 7(?) ?–?. Printed in the United Kingdom
© 2001 Cambridge University Press

Applied morphological processing of English

GUIDO MINNEN†
JOHN CARROLL
DARREN PEARCE
Cognitive and Computing Sciences
University of Sussex
Brighton BN1 9QH, UK
email: {firstname.lastname}@cogs.susx.ac.uk

(*Received 26 January 2001; revised 3 May 2001*)

Abstract

We describe two newly developed computational tools for morphological processing: a program for analysis of English inflectional morphology, and a morphological generator, automatically derived from the analyser. The tools are fast, being based on finite-state techniques, have wide coverage, incorporating data from various corpora and machine readable dictionaries, and are robust, in that they are able to deal effectively with unknown words. The tools are freely available. We evaluate the accuracy and speed of both tools and discuss a number of practical applications in which they have been put to use.

1 Introduction

Morphological processing is a core component in many different types of language-based computer applications. We describe two newly developed tools for robust processing of English inflectional morphology that can be used in such applications: a morphological analyser and a morphological generator.[1]

The morphological analyser maps a word form to its underlying lemma plus inflectional suffix (if any) using knowledge about the morphology of English acquired semi-automatically from several large corpora and machine readable dictionaries. The analyser is based on one originally developed as part of the GATE information extraction framework (Cunningham, Wilks, & Gaizauskas, 1996). The morphological generator is derived automatically from the analyser by means of a compilation process. This means that any changes and improvements to the analyser are reflected in the generator with no manual effort. The generator synthesises a word form given a lemma, its part-of-speech (PoS), and the type of inflection required.

† Current address: Motorola Human Interface Laboratory, Schaumburg, IL 60196, USA; email: minnen@labs.mot.com
[1] Minnen, Carroll, and Pearce (2000) present some of the aspects of this work that relate to generation.

Neither the analyser nor the generator contain an explicit lexicon or word-list, but instead comprise a set of morphological generalisations together with a list of exceptions for specific (irregular) word forms. This organisation into generalisations and exceptions can save time and effort in system development since the addition of new vocabulary with regular morphology does not require any changes to the source files. In addition, for generation, the generalisation-exception architecture can be used to specify—and also override—preferences in cases where a lemma has more than one possible surface word form given a particular inflectional type and part-of-speech.

The morphological tools are easy to integrate into applications, each being packaged up as a Unix 'filter'. The tools are based on efficient finite-state techniques, and are implemented using the widely-available Flex utility (a reimplementation of the AT&T Unix Lex tool (Levine, Mason, & Brown, 1992)). We evaluate the accuracy and the speed of both tools using the CELEX lexical database of English (Baayen, Piepenbrock, & van Rijn, 1993) and the British National Corpus (BNC; Burnard, 1995).

The analyser and the generator are freely available to the NLP research community (see section 7 below), and are currently being applied in a number of practical NLP systems. In this article we describe in detail one such use (of both tools), in a prototype system for automatic simplification of English newspaper text. This application gives rise to a number of practical morphological and orthographic issues which we also discuss.

The article is structured as follows. Sections 2 and 3 describe the morphological analyser and generator, respectively. Section 4 evaluates both tools. Section 5 discusses a number of practical applications in which the morphological processing tools have been put to use. Section 6 relates our work to that of others, and we conclude (section 7) with directions for future work.

2 Morphological analysis

Morphological analysis is a vital processing step in many NLP systems. It maps word forms into lemmas together with any morphosyntactic information conveyed by affixes. This enables a system, for instance, to store information about words keyed by lemma rather than by full word form, to use the morphosyntactic information to guide subsequent linguistic processing, and to use words' lemmas to generalise across the corresponding word form tokens.

A number of wide-coverage morphological analysis systems for English have been developed over the years. These differ along a number of dimensions. Firstly, the processing techniques used include finite-state transduction, finite-state word segmentation coupled with a morpheme-level grammar, and implementations of theories of lexical inheritance. Depending on the technique used, the systems might or might not be efficient enough to process large amounts of text. Secondly, in some analysers the lexical data consist of a list of lemmas and information about their inflectional properties, whereas in others it comprises an exhaustive list of inflected word forms. The data or the systems (or both) may be freely available, or only obtainable under restrictive licensing conditions. Finally, some systems are able to deal with morphological generation within a single framework, whereas others are inherently non-reversible. In Section 6 we present a detailed review of extant approaches and contrast them with ours.

Our tools cover the inflectional morphology of English nouns and verbs, handling the productive English suffix *s* for the plural form of nouns and the third person singular present tense of verbs, and *ed* for the past tense, *en* for the past participle, and *ing* for the present participle forms of verbs.[2]

By default, the input to the analyser is assumed to be tagged, using PoS labels that follow the same pattern as the Lancaster CLAWS tag sets (Garside, Leech, & Sampson, 1987; Burnard, 1995)—with verb tags starting with *V* etc. The analyser returns the lemma of the word form, a specification of the inflectional type, and the PoS label.[3] Similarly, the input to the generator is a lemma, an inflectional type, and a PoS label. When processing languages with richer inflectional morphology than English, performing morphological analysis *before* PoS determination may be more appropriate. However, in expecting tagged input, the analyser conforms to current established practice in large-scale, applied processing of English text (e.g. Baldwin et al., 1995; Gaizauskas, Wakao, Humphreys, Cunningham, & Wilks, 1995), in which a PoS tagger incorporating a large full-form lexicon (often acquired from large training corpora) and unknown word guesser is applied first, with morphological analysis carried out in a subsequent phase.

2.1 Morphological analysis using Flex

The morphological analyser is implemented in Flex. Flex is conventionally used for constructing 'scanners': programs that recognise lexical patterns in textual input (Levine et al., 1992). A Flex description—the high-level description of a scanner that Flex takes as input—consists of a set of 'rules': pairs of regular expression patterns (which Flex compiles into deterministic finite-state automata (Aho, Sethi, & Ullman, 1986)) and actions consisting of arbitrary C code. Flex creates as output a C program which at run-time scans a text looking for occurrences of the regular expressions. Whenever it finds one, it executes the corresponding C code. Flex is part of the Berkeley Unix distribution, and as a result Flex programs are highly portable. The standard version of Flex works with all of the ISO-8559 character sets; Unicode support is also available.

The analyser comprises: (1) a set of of approximately 1,400 Flex rules which express morphological generalisations together with a list of exceptions for specific words; and (2) around 350 lines of C/Flex code which, among other things, defines the functions called in the actions specified in those rules. The input is expected to be a sequence of tokens of the form *wordform_label*, where *wordform* specifies the

[2] At present, we do not cover comparative and superlative forms of adjectives since their productivity is much less predictable.

[3] The morphological analyser should therefore be distinguished from a stemmer (e.g. Porter, 1980), which heuristically *truncates* word forms.

word form to be analysed, and *label* specifies its PoS label, the symbol _ being a delimiter. Input that is not of this form is copied through to the output unchanged.

Example (1) illustrates a Flex rule that captures a morphological generalisation.[4]

```
(1)          {A}+{C}"ied"      {return(lemma(3,"y","ed"));}
```

The left-hand side of the rule is a regular expression. `{A}` stands for exactly one occurrence of an element of the (predefined) character set `A`, and `{A}+`, a sequence of one or more `A`s. We assume here that `A` abbreviates the (upper and lower case) letters of the alphabet, and `C` the consonants. Double quotes indicate literal character symbols. The right-hand side of the rule gives the code to be executed when an input string matches the regular expression. When the rule matches the input *carried*, for example, the function `lemma` (defined elsewhere in the analyser) is called to determine the word form corresponding to the input: in this case the function removes the last three characters of the word form, and then attaches the character *y*, the delimiter *+*, and the inflection type *ed*; the output is therefore *carry+ed*.

Of course not all past tense/participle verb forms ending in *ied* are analysed correctly by the simple rule in (1) since English contains many irregularities and subregularities. These are dealt with using additional, more specific, rules. The order in which these rules are applied to the input follows exactly the order in which the rules appear in the Flex description. This makes for a very simple and perspicuous way to express generalizations and exceptions. For instance, the rule in (2) analyses the verb *boogied* as *boogie+ed* rather than the incorrect **boogy+ed* which would follow from the application of the more general rule in (1).

```
(2)            "boogied"     {return(lemma(1,"","ed"));}
```

The organisation into generalisations and exceptions as described in this section can save time and effort in system development. Dealing with a new item of vocabulary does not necessarily require that the analyser source be modified; if the word has regular morphology then no change is needed, otherwise a single new rule implementing the irregularity must be added. This can lead to a considerable saving in development time. Moreover, words that have not previously been encountered are very often handled correctly by the rules expressing generalisations.

2.2 Processing options

The morphological analyser supports various Unix command line options to give the user more flexibility. One option indicates that no PoS labels are present in the input; although this can be useful when no tagger is available, it seriously compromises accuracy. Another option controls whether PoS labels in the input are copied through to the output. In addition, for development purposes, there is an

[4] The rules used in the examples throughout the remainder of the article have been simplified slightly for reasons of exposition. In particular, the treatment of the delimiter symbol _ and the PoS label is ignored.

Original sentence:
But_CJC he_PNP was_VBD also_AV0 drawn_VVN to_PRP the_AT0 Jews_NN2 whom_PNQ he_PNP met_VVD in_PRP their_DPS Polish_AJ0 villages_NN2 ,_, victims_NN2 of_PRF persecution_NN1 and_CJC war_NN1 ,_, &bquo_“ old_AJ0 Jews_NN2 with_PRP prophets_NN2 &equo_” beards_NN2 and_CJC passionate_AJ0 rags_NN2 ,_, to_PRP their_DPS ruined_AJ0 ghettoes_NN2 and_CJC synagogues_NN2 ._.

Analysed sentence:
But_CJC he_PNP be+ed_VBD also_AV0 draw+en_VVN to_PRP the_AT0 Jew+s_NN2 whom_PNQ he_PNP meet+ed_VVD in_PRP their_DPS Polish_AJ0 village+s_NN2 ,_, victim+s_NN2 of_PRF persecution_NN1 and_CJC war_NN1 ,_, &bquo_“ old_AJ0 Jew+s_NN2 with_PRP prophet+s_NN2 &equo_” beard+s_NN2 and_CJC passionate_AJ0 rag+s_NN2 ,_, to_PRP their_DPS ruined_AJ0 ghetto+s_NN2 and_CJC synagogue+s_NN2 ._.

Fig. 1. An example of morphological analysis.

interactive mode that allows the user to provide input via the command line rather than a file.

Figure 1 gives an example of morphological analysis of a sentence from the BNC.

2.3 Incorporated data

The original GATE morphological analyser (Cunningham et al., 1996) included exception lists taken from the WordNet 1.5 lexical database (Fellbaum, 1998), containing 5,254 verbs and 5,973 nouns. Of these exceptions, around 75% were covered by a set of approximately fifty regular expression rules.[5] In addition to the WordNet data, we have semi-automatically acquired new rules for irregular and subregular words from the following corpora and machine readable dictionaries: the LOB corpus (Garside et al., 1987), the University of Pennsylvania Treebank (Marcus, Santorini, & Marcinkiewicz, 1993), the SUSANNE corpus (Sampson, 1995), the Lancaster/IBM Spoken English Corpus (Knowles, Williams, & Taylor, 1996), the Oxford Psycholinguistic Database (Quinlan, 1992), and the 'Computer-Usable' OALDCE from the Oxford Text Archive (Mitton, 1992). We have intentionally avoided using proprietary data—for example from commercially published dictionaries—to keep the tools free of restrictive licensing conditions.

3 Morphological generation

Most approaches to natural language generation (NLG) ignore morphological variation during word choice and application of grammatical constraints, postponing the computation of the actual output word forms to a final stage, sometimes termed

[5] WordNet entries including the _ character, entries with alternative forms, and obvious errors were all removed, leaving around 19% of the original WordNet exceptions.

'linearisation'. An important advantage of this setup is that the preceding syntactic/lexical realisation component does not have to consider all possible word forms corresponding to each lemma (Shieber, van Noord, Moore, & Pereira, 1990). For development and maintenance reasons it is also advantageous for morphological generation to be a postprocessing component that is separate from the rest of the NLG system. Moreover, such a generator can be used on its own in other types of applications that produce natural language but do not contain a standard realisation component, such as text simplification (see section 5).

The Flex description of the morphological generator is derived automatically from the analyser through a compilation process which is computationally very cheap (taking just a few seconds). A benefit of this arrangement is that after modifications to the analyser, the generator can be updated automatically and will reflect the modifications without any further manual effort. Just like the analyser, it supports various command line options and an interactive mode.

The input to the generator is expected to be a sequence of tokens of the form *lemma+inflection_label*, where *lemma* specifies the lemma of the word form to be generated, *inflection* specifies the type of inflection (i.e. *s*, *ed*, *en* or *ing*), and *label* specifies the PoS of the word form; the symbols + and _ are delimiters.

3.1 Deriving the generator

The compilation process 'inverts' each analyser regular expression pattern / action pair to derive the generator. It does this by simulating the effect of the analyser action on the pattern; this produces the new generator pattern. The new action consists of a call to a function that removes the last n characters from the input, where n is the number of characters the analyser action adds, and then appends the characters that were removed by the analyser action. So, for example, (3) shows the compilation process applied to the rule in (1).

```
(3)   {A}+{C}"ied"     {return(lemma(3,"y","ed"));}     (analyser)
                                   ↓
      {A}+{C}"y+ed"    {return(glemma(4,"ied"));}       (generator)
```

In addition to the processing of individual rules, the compiler concatenates to the generator description the required C/Flex declarations and definitions of the generator actions.

More formally, given the analyser action `lemma(`m`,`s_1`,`s_2`)`, the corresponding generator action is `glemma(`n`,`s_3`)`, where n is the sum of $length(s_1)$ and $length(s_2)$ plus 1 (for the + delimiter), and s_3 is the last m characters of the analyser pattern. Thus for the rule analysing the irregular plural *crises* of the noun *crisis* in (4), we have $m = 2$, s_1 = `"is"`, and s_2 = `"s"`.

```
(4)          "crises"     {return(lemma(2,"is","s"));}
```

The compilation process produces the generator rule in (5), in which the call to `glemma` deletes the last $n = 4$ characters of the input string (i.e. `"is+s"`), and then appends s_3 = `"es"` (the last m characters of the analyser pattern).

(5)
```
"crisis+s"     {return(glemma(4,"es"));}
```

3.2 Inflectional preferences

Flex does not require the rules constituting a scanner to be mutually exclusive. Thus, for morphological generation, the description can encode the inflectional morphology of lemmas that have more than one possible inflected form given a particular PoS label and inflectional type. An example of this is the multiple inflections of the noun *cactus*, which has not only the Latinate plural form *cacti* but also the Anglicised plural form *cactuses*. In addition, inflections of some words differ according to dialect. For example, the past participle form of the verb *to bear* is *borne* in British English, whereas in American English the preferred word form is *born*.

In cases where there is more than one possible inflection for a particular input lemma, the order of the rules in the description determines the word form output. For example, with the noun *cactus*, a preference for generating *cacti* is expressed by having the corresponding rule precede the one generating *cactuses*.

In Flex, rule choice based on ordering is overridden if the the second or subsequent match covers a longer segment of the input: in that case the longest match heuristic applies (Levine et al., 1992). This cannot happen in the case of the generator, though, since the input is matched and processed word-by-word. However, it should be noted that the generator will always choose between multiple inflections: there is no way for it to execute all relevant actions and output all possible word forms for a particular input.[6]

A related issue concerns the treatment of past tense and past participle forms that are identical. In such cases, we decided the analyser should output the lemma plus (only) the suffix *+ed*. However, the generator must be able to accept the *+en* suffix specification for these words, so for each lemma concerned we include a special comment in the analyser source telling the compilation program that builds the generator to allow *+en* as well as *+ed*.

3.3 Consonant doubling

Another issue that poses problems for morphological generation is the phenomenon of consonant doubling. This occurs mainly in British English, and involves the doubling of the final consonant of a verb lemma in the participle and past tense forms. For example, the past tense/participle inflection of the verb *travel* is *travelled* in British English, where the final consonant of the lemma is doubled before the suffix is attached. In American English the corresponding form is usually *traveled*.

[6] In the implementation, rules corresponding to dispreferred word forms are actually created in a commented out form by the analyser-to-generator compilation process, to avoid Flex warnings saying that the rule patterns cannot be matched. We indicate this to the compilation program via a special comment on each such rule in the analyser source.

Consonant doubling is triggered on the basis of both orthographic and phonological information: when a word ends in one vowel followed by one consonant and the last part of the word is stressed, the consonant is usually doubled (Procter, 1995). However, there are exceptions to this, and in any case the input to the morphological generator does not contain information about stress.

Consider the Flex rule in (6).

```
(6) {A}+"t+ed_V"      {return(glemma("t","ed"));}
```

Given the input *submit+ed_V* this rule correctly generates *submitted* by adding *t* and then *ed* to the lemma *submit*. However, the verb *to exhibit* does not undergo consonant doubling so this rule would generate, incorrectly, the word form *exhibitted*.

In order to ensure that the correct inflection of a verb is generated, the morphological generator uses a list of around 1,100 lemmas that allow consonant doubling which we have extracted automatically from the BNC. The list is checked before inflecting a verb. We found that there are many more verbs that do not allow consonant doubling, so listing the verbs that do is the most economical solution. An added benefit is that if a lemma *does* allow consonant doubling but is not present in the list then the word form generated will still be correct with respect to American English.

4 Evaluation

In order to evaluate the accuracy of both tools, we extracted from the CELEX lexical database of English (version 2.5) all inflected noun and verb word forms, i.e. past tense, past and present participle, and third person singular present tense inflections of verbs, and all plural nouns. After excluding multi-word entries (phrasal verbs, etc.) we ended up with 38,882 word forms (out of the total of 160,595 entries in CELEX). We input each of these word forms to the analyser and compared the output with the lemmatisation provided by CELEX. This revealed 895 mistakes apparently made by the analyser. In a number of cases the CELEX analysis was wrong in that it disagreed with the entry for the word in the *Cambridge International Dictionary of English* (Procter, 1995). We ignored these mistakes. Of the remaining mistakes a total of 396 concerned word forms that do not occur anywhere in the 100M words of the BNC; we categorised these as irrelevant for practical applications and so discarded them. Thus the *type* accuracy of the morphological analyser with respect to CELEX was 99.0%. The *token* accuracy was 99.7% with respect to the 14,825,661 relevant tokens in the BNC. Errors made by the analyser consisted mainly of cases where a rule was applied that was too general. For example, the third person singular present tense inflection of verbs like *dispense*, *incense*, and *license* were incorrectly analysed as having a lemma ending in *s*, as in *dispens+s*.

We used the same methodology to evaluate the morphological generator, except that this time the input was the correct analyses and we compared the output to the corresponding word forms. Numerically, the results were similar: the generator made 979 mistakes, of which 346 concerned word forms not occurring in the BNC.

Table 1. *Accuracy and speed of the analyser and generator.*

	type accuracy	token accuracy	inflections (words/sec)	throughput (words/sec)
Analyser	99·94%	99·93%	174K	240K
Generator	99·96%	99·98%	148K	218K

After discarding these, the morphological generator had 98.4% type accuracy with respect to CELEX, and 99.4% token accuracy with respect to the BNC.

In the course of the evaluation we noticed that CELEX is inconsistent with respect to word forms exhibiting consonant doubling. In the main, it contains (British English) consonant doubled forms. However, for 129 lemmas in CELEX, the BNC contains consonant doubled forms and CELEX does not. For example, the BNC contains the form *programming* but CELEX does not. The form *programing* does occur in CELEX but not in the BNC. If these cases—which make up more than 20% by type of the remaining mistakes made by the generator—are taken into account then the accuracy results improve: the type accuracy goes up to 98.7% and the token accuracy to 99.5%.

After this evaluation we collected up the errors and we have now fixed almost all of them. The error rate of the analyser and generator is now in the range four to six errors per ten thousand verb/noun types, and between two and seven errors per ten thousand tokens; Table 1 summarises the current accuracy figures. We have also tested the processing speed of the tools. Despite their wide coverage the morphological analyser and generator are very fast: they analyse/generate inflected verbs and nouns at a rate of more than 140K per second (as measured on a Sun Ultra 10 360MHz workstation). When applied to ordinary running text both rates are in excess of 210K words per second (Table 1). The executable files are relatively compact, each being approximately 700Kbytes in size.

5 The morphological analyser and generator in applied systems

Both morphological tools are packaged up as separate, individual modules, with the intention that they be treated by application systems as 'black boxes'. There are a number of advantages in localising morphological processing in this way. For the tool developer, a benefit is that since there are no competing claims on the representation framework from other types of linguistic and non-linguistic knowledge, the developer is free to express morphological information in a perspicuous and elegant manner. For the application developer, such a setup facilitates more systematic and reliable updating. From a software engineering perspective, modularisation reduces system development costs and increases system reliability. Moreover, as standalone, independent modules, the morphological tools are more easily shareable between different NLP applications, and integrated into new ones.

The tools are being used in a number of practical applications by various groups in the UK. The morphological analyser has been used to group together inflectional variants of verbs and the heads of their complements for unsupervised learning of verb subcategorisation (Briscoe & Carroll, 1997), selectional preferences (McCarthy, 1997), and diathesis alternations (McCarthy & Korhonen, 1998). The analyser was used to create the frequency lists for the sampling frame for the SENSEVAL-1 exercise (Kilgarriff, 1998). It forms part of a prototype word sense disambiguation system (Carroll & McCarthy, 2000), and is being used in research into methods for the semantic interpretation of complex nominals in medical domains (Grover & Lascarides, 2001). Other, commercial applications are in corpus processing for dictionary publishing, in particular as part of the creation of 'word sketches' for lexicographers (Kilgarriff & Rundell, 1999), and in product marketing, producing lemma frequency data from large corpora to inform the selection of new brand names. An earlier version of the analyser is used in the LaSIE system (Gaizauskas et al., 1995) for robust lemmatisation in information extraction tasks.

These applications have run the tools using a variety of methods. The tools have been invoked as standalone programs, run within an XML processing pipeline and as a module in a TIPSTER-style document processing architecture (inside appropriate 'wrapper' programs), and called (with piped input and output) from Perl and Lisp programs.

The analyser and the generator are also being used by students on NLP courses at a number of universities in Europe.

5.1 Case study: text simplification

In addition to the uses outlined above, the morphological analyser and generator both form part of a prototype system for automatic simplification of English newspaper text (Carroll et al., 1999). The goal is to help people with aphasia (a language impairment typically occurring as a result of a stroke or head injury) to better understand news stories published in their local newspaper. The system comprises two main components: an analysis module which downloads the source newspaper texts from the World Wide Web and computes syntactic analyses for the sentences in them, and a simplification module which operates on the output of the analyser to improve the comprehensibility of the text. Syntactic simplification (Canning & Tait, 1999) transforms the syntax trees produced in the analysis phase, for example converting sentences in the passive voice to active, and splitting long sentences at appropriate points. A subsequent lexical simplification stage (Devlin & Tait, 1998) replaces difficult or rare content words with more common synonyms.

The analysis component uses the morphological analyser to enable the base forms of words to be passed through the system; this eases the task of the simplification module as it does not need to consider all possible inflections of the lemmas in the text. The final processing stage in the system is therefore morphological generation, using the generator described in the previous section. This is illustrated in figure 2.

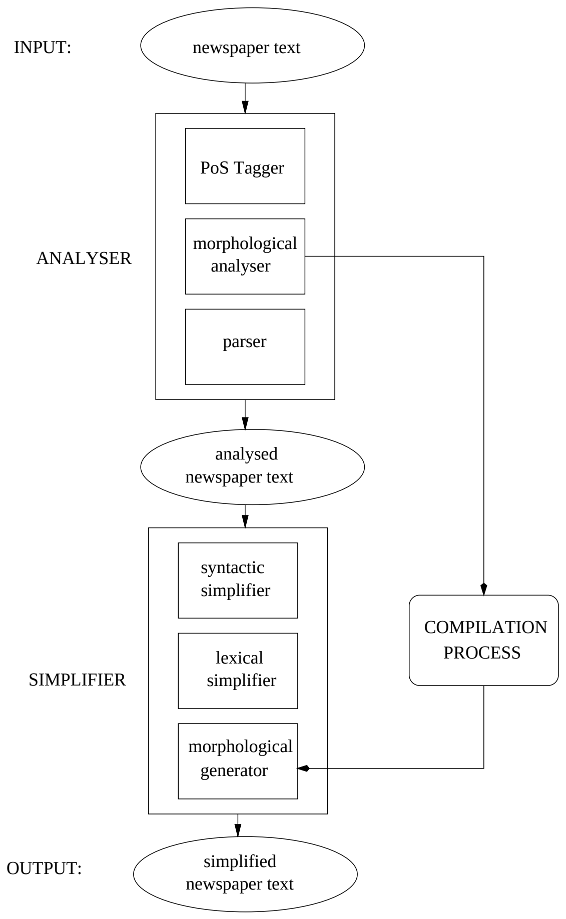

Fig. 2. Text simplification system architecture.

5.1.1 Morphological processing

We have tested the components of the simplification system on a corpus of a thousand news stories downloaded from the *Sunderland Echo* (a local daily newspaper in North-East England). In our testing we found that newly encountered vocabulary only rarely necessitates modification of the analyser/generator source; if the word has regular morphology then it is handled by the rules expressing generalisations. Also, a side-effect of the fact that the generator is derived from the analyser is that the two modules have exactly the same coverage and are guaranteed to stay in step with each other. This is important in the context of an applied system. The accuracy of the generator is quite sufficient for this application; our experience is that typographical mistakes in the original newspaper text are much more common than errors in morphological processing.

5.1.2 Orthographic processing

Some orthographic phenomena span more than one word. Such phenomena cannot be dealt with as part of morphological generation since this works strictly on individual words. We have therefore implemented a final orthographic postprocessing stage. Consider the sentence:[7]

(7) **Brian Cookman is the attraction at the* ***King 's Arms*** *on Saturday night and* ***he will*** *be back on Sunday night for* ***a acoustic*** *jam session.*

This is incorrect orthographically because the determiner in the final noun phrase should be *an*, as in *an acoustic jam session.* In fact *an* must be used if the following word starts with a vowel sound, and *a* otherwise. We achieve this, again using a filter implemented in Flex, with a set of general rules keying off the next word's first letter (having skipped any intervening sentence-internal punctuation), together with a list of exceptions (for example, *heir*, *unanimous*) collected using the pronunciation information in the 'Computer-Usable' OALDCE, supplemented by further cases (for example, *unidimensional*) found in the BNC. In the case of abbreviations or acronyms (recognised by the occurrence of non-word-initial capital letters and trailing full-stops) we key off the pronunciation of the first letter considered in isolation.

Similarly, the orthography of the genitive marker cannot be determined without taking context into account, since it depends on the identity of the last letter of the preceding word. In the sentence in (7) we need only eliminate the space before the genitive marking, obtaining *King's Arms.* But, following the newspaper style guide, if the preceding word ends in *s* or *z* we have to 'reduce' the marker as in, for example, *Stacey Edwards' skilful fingers.*

The generation of contractions presents more of a problem. For example, changing *he will* to *he'll* would make (7) more idiomatic. But there are cases where this type of contraction is not permissible. Since these cases seem to be dependent on syntactic context (see section 6 below), and we have syntactic structure from the analysis phase, we are in a good position to make the correct choice. However, we have not yet tackled this issue and currently take the conservative approach of not contracting in any circumstances.

6 Discussion

We are following a well-established line of work using finite-state techniques for lexical and shallow syntactic NLP tasks (e.g. Karttunen, Chanod, Grefenstette, & Schiller, 1996). Lexical transducers have been used extensively for morphological analysis, and in theory a finite-state transducer implementing an analyser can be reversed to produce a generator. However, we are not aware of published research on finite-state morphological generators that: (1) establishes whether in practice

[7] This sentence is taken from the news story "The demise of Sunderland's Vaux Breweries is giving local musicians a case of the blues" published in the *Sunderland Echo* on 26 August 1999.

they perform with similar efficiency to morphological analysers; (2) quantifies their type/token accuracy with respect to an independent, extensive 'gold standard'; or (3) discusses the issues involved when integrating them into larger, applied systems. Furthermore, although a number of finite-state compilation toolkits (e.g. Karttunen, 1994) are publicly available or can be licensed on reasonable terms for research use, associated wide-coverage linguistic descriptions—for example English morphological lexicons—are usually commercial products targeted at technology provider companies and are therefore beyond the reach of most of the NLP research community.

The work reported here is also related to work on lexicon representation and morphological processing using the DATR representation language (Cahill, 1993; Evans & Gazdar, 1996). However, we adopt less of a theoretical and more of an engineering perspective, focusing on morphological processing in the context of wide-coverage practical NLP applications. There are also parallels to research in the two-level morphology framework (Koskenniemi, 1983), although in contrast to our approach this framework has required exhaustive lexica and hand-crafted morphological (unification) grammars in addition to orthographic descriptions (van Noord, 1991; Ritchie, Russell, Black, & Pulman, 1992). The SRI Core Language Engine (Alshawi, 1992) uses a set of declarative segmentation rules which are similar in content to our rules and are used in reverse to generate word forms. The SRI system, however, is not publicly available, again requires an exhaustive stem lexicon, and the rules are not compiled into an efficiently executable finite-state machine but are only interpreted.

The work that is perhaps the most similar in spirit to ours is that of the LADL group at the University of Paris 7, in their compilation of large lexicons of inflected word forms into finite-state transducers (Mohri, 1996). The resulting analysers run at a comparable speed to our tools and the (compacted) executables are of similar size. However, a full-form lexicon is unwieldy and inconvenient to update, and a system derived from it cannot cope gracefully with unknown words because it does not contain generalisations about regular or subregular morphological behaviour. Also, the LADL data is not freely available.

The only other large-scale freely available morphological processing system of which we are aware (Karp, Schabes, Zaidel, & Engedi, 1992) comprises a morphological analyser, but not a generator. The system uses an exhaustive lexicon and so again cannot deal with words not listed in the lexicon. However, the system can accurately analyse text that has not been pre-tagged with PoS labels, whereas our analysis tool cannot. It also has the advantage of being able to provide more than a single analysis for homographs such as the verb *lay* (which can be either the base form of *to lay* or the past tense of *to lie*). On the other hand, Karp et al.'s system stores its lexical data in a disk-based hash table so disk seek times would limit the maximum rate of processing to only around 2K words per second.

Turning to morphological generation, the morphological components of current widely-used NL realisers (or *tactical generation* systems) tend to consist of hard-wired procedural code that is tightly bound to the workings of the rest of the system. For instance, the Nigel grammar (Matthiessen, 1984) contains Lisp code that classifies verb, noun and adjective endings, and these classes are picked up by further code inside the KPML system (Bateman, 2000) itself which performs inflectional generation by stripping off variable length trailing strings and concatenating suffixes. All morphologically subregular forms must be entered explicitly in the lexicon, as well as irregular ones. The situation is similar in FUF/SURGE, morphological generation in the SURGE grammar (Elhadad & Robin, 1996) being performed by procedures which inspect lemma endings, strip off trailing strings when appropriate, and concatenate suffixes.

In current NLG systems, orthographic information is distributed throughout the lexicon and is applied via the grammar or by hard-wired code. This makes orthographic processing difficult to decouple from the rest of the system, compromising maintainability and ease of reuse. For example, in SURGE, a marker for *a/an* usage can be added to a lexical entry to indicate that the word's initial sound is consonant- or vowel-like and is contrary to what its orthography would suggest. The appropriate indefinite article is inserted by procedures associated with the grammar. In DRAFTER-2 (Power, Scott, & Evans, 1998), an *a/an* feature can be associated with a lexical entry, and its value is propagated up to the noun phrase level through leftmost rule daughters in the grammar (Power, personal communication). Both of these systems interleave orthographic processing with other processes in realisation. In addition, neither has a mechanism for stating exceptions for whole subclasses of words, for example those starting *us* followed by a vowel—such as *use* and *usual*—which must be preceded by *a*.

We are not aware of any literature describing (practical) NLG systems that generate contractions. However, interesting linguistic research in this direction is reported by Pullum and Zwicky (1997). This work investigates the underlying syntactic structure of sentences that block auxiliary reductions, for example those with VP ellipsis as in (8).

(8) **She's usually home when he's.*

7 Conclusions

We have described two tools for fast and accurate processing of English inflectional morphology. The main features of these tools are:

wide coverage and high accuracy They incorporate data from several large corpora and machine readable dictionaries. An evaluation has shown the error rate to be very low.

robustness The tools do not contain an explicit lexicon or word-list, but instead comprise a set of morphological generalisations together with a list of exceptions for specific (irregular) words. Unknown words are very often handled correctly by the generalisations.

maintainability and ease of use The organisation into generalisations and exceptions can save development time since addition of new vocabulary that has regular morphology does not require any changes to be made. The tools are packaged up as Unix filters, and have been integrated into applications using a number of different methods.

speed and portability The tools are based on efficient finite-state techniques, and implemented using the widely available Flex utility.

freely available The morphological processing tools and orthographic post-processor are free for academic or industrial research use; the sources (and pre-built binaries for the Solaris operating system) can be downloaded via <http://www.cogs.susx.ac.uk/lab/nlp/carroll/morph.html>.

We are currently working on extending the tools to cover comparative and superlative forms of adjectives. In the future we intend to investigate the use of phonological information in machine readable dictionaries for a more principled solution to the consonant doubling problem. We also plan to further increase the flexibility of the generator by including an option that allows the user to choose whether it has a preference for generating British or American English spelling.

Acknowledgements

This work was funded by UK EPSRC project GR/L53175 'PSET: Practical Simplification of English Text', and by an EPSRC Advanced Fellowship to the second author. The original version of the morphological analyser was developed by Kevin Humphreys and kindly made available to us by the University of Sheffield GATE project. Chris Brew, Bill Fisher, Dale Gerdemann, Gerald Gazdar, Adam Kilgarriff, Diana McCarthy, and Ehud Reiter have either suggested or contributed improvements to the analyser/generator.

References

Aho, A., Sethi, R., & Ullman, J. (1986). *Compilers: principles, techniques and tools.* Reading, MA: Addison-Wesley.

Alshawi, H. (Ed.). (1992). *The Core Language Engine.* Cambridge, MA: MIT Press.

Baayen, H., Piepenbrock, R., & van Rijn, H. (1993). *The CELEX lexical database (CD-ROM).* University of Pennsylvania, Philadelphia, PA: Linguistic Data Consortium.

Baldwin, B., Reynar, J., Collins, M., Eisner, J., Ratnaparkhi, A., Rosenzweig, J., Sarkar, A., & Srinivas, B. (1995). University of Pennsylvania: description of the University of Pennsylvania system used for MUC-6. In *Proceedings of the Sixth Message Understanding Conference.* San Francisco, CA: Morgan Kaufmann.

Bateman, J. (2000). *KPML (version 3.1) March 2000.* University of Bremen, Germany. (<http://www.fb10.uni-bremen.de/anglistik/langpro/kpml/README.html>)

Briscoe, E., & Carroll, J. (1997). Automatic extraction of subcategorization from corpora. In *Proceedings of the Fifth ACL Conference on Applied Natural Language Processing* (pp. 356–363). Washington, DC.

Burnard, L. (1995). *Users reference guide for the British National Corpus* (Tech. Rep.). Oxford University Computing Services, UK.

Cahill, L. (1993). Morphonology in the lexicon. In *Proceedings of the Sixth Conference of the European Chapter of the Association for Computational Linguistics* (pp. 87–96). Utrecht, The Netherlands.

Canning, Y., & Tait, J. (1999). Syntactic simplification of newspaper text for aphasic readers. In *Proceedings of the ACM SIGIR Workshop on Customised Information Delivery.* Berkeley, CA.

Carroll, J., & McCarthy, D. (2000). Word sense disambiguation using automatically acquired verbal preferences. *Computers and the Humanities*, *34*(1–2), 109–114.

Carroll, J., Minnen, G., Pearce, D., Canning, Y., Devlin, S., & Tait, J. (1999). Simplifying English text for language impaired readers. In *Proceedings of the Ninth Conference of the European Chapter of the Association for Computational Linguistics* (pp. 269–270). Bergen, Norway.

Cunningham, H., Wilks, Y., & Gaizauskas, R. (1996). GATE — a general architecture for text engineering. In *Proceedings of the 16th Conference on Computational Linguistics* (pp. 1057–1060). Copenhagen, Denmark.

Devlin, S., & Tait, J. (1998). The use of a psycholinguistic database in the simplification of text for aphasic readers. In J. Nerbonne (Ed.), *Linguistic Databases* (pp. 161–173). Stanford, CA: CSLI Publications.

Elhadad, M., & Robin, J. (1996). *An overview of SURGE: a reusable comprehensive syntactic realization component* (Tech. Rep. No. 96-03). Dept of Mathematics and Computer Science, Ben Gurion University, Israel.

Evans, R., & Gazdar, G. (1996). DATR: a language for lexical knowledge representation. *Computational Linguistics*, *22*(2), 167–216.

Fellbaum, C. (Ed.). (1998). *WordNet: an electronic lexical database.* Cambridge, MA: MIT Press.

Gaizauskas, R., Wakao, T., Humphreys, K., Cunningham, H., & Wilks, Y. (1995). University of Sheffield: description of the LaSIE system as used for MUC-6. In *Proceedings of the Sixth Message Understanding Conference.* San Francisco, CA: Morgan Kaufmann.

Garside, R., Leech, G., & Sampson, G. (1987). *The computational analysis of English: a corpus-based approach.* London, UK: Longman.

Grover, C., & Lascarides, A. (2001). XML-based data preparation for robust deep parsing. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics.* Toulouse, France.

Karp, D., Schabes, Y., Zaidel, M., & Engedi, D. (1992). A freely available wide coverage morphological analyser for English. In *Proceedings of the 14th International Conference on Computational Linguistics* (pp. 950–955). Nantes, France.

Karttunen, L. (1994). Constructing lexical transducers. In *Proceedings of the 15th International Conference on Computational Linguistics* (pp. 406–411). Kyoto, Japan.

Karttunen, L., Chanod, J.-P., Grefenstette, G., & Schiller, A. (1996). Regular expressions for language engineering. *Natural Language Engineering*, *2*(4), 305–329.

Kilgarriff, A. (1998). Gold standard datasets for evaluating word sense disambiguation programs. *Computer Speech and Language*, *12*(4), 453–472.

Kilgarriff, A., & Rundell, M. (1999). *Lexicography for computationalists.* Tutorial given at the 37th Annual Meeting of the Association for Computational Linguistics. College Park, MD. (<http://www.itri.bton.ac.uk/~Adam.Kilgarriff/wordsketches.html>)

Knowles, G., Williams, B., & Taylor, L. (Eds.). (1996). *A corpus of formal British English speech. The Lancaster/IBM Spoken-English Corpus.* London, UK: Longman.

Koskenniemi, K. (1983). Two-level model for morphological analysis. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence* (pp. 683–685). Karlsruhe, Germany.

Levine, J., Mason, T., & Brown, D. (1992). *Lex & Yacc* (Second ed.). Sebastopol, CA: O'Reilly & Associates.

Marcus, M., Santorini, B., & Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, *19*(2), 313–330.

Matthiessen, C. (1984). Systemic Grammar in computation: the Nigel case. In *Proceedings of the First Conference of the European Chapter of the Association for Computational Linguistics* (pp. 155–164). Pisa, Italy.

McCarthy, D. (1997). Word sense disambiguation for acquisition of selectional preferences. In *Proceedings of the ACL/EACL'97 Workshop on Automatic Information Extraction and Building of Lexical Semantic Resources for NLP Applications* (pp. 52–61). Madrid, Spain.

McCarthy, D., & Korhonen, A. (1998). Detecting verbal participation in diathesis alternations. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics* (pp. 1493–1495). Montreal, Canada.

Minnen, G., Carroll, J., & Pearce, D. (2000). Robust, applied morphological generation. In *Proceedings of the First International Conference on Natural Language Generation* (pp. 201–208). Mitzpe Ramon, Israel.

Mitton, R. (1992). *A description of a computer-usable dictionary file based on the Oxford Advanced Learner's Dictionary of Current English.* (<ftp://ota.ox.ac.uk/pub/ota/public/dicts/710/text710.doc>)

Mohri, M. (1996). On some applications of finite-state automata theory to natural language processing. *Natural Language Engineering*, *2*(1), 61–80.

Porter, M. (1980). An algorithm for suffix stripping. *Program*, *14*, 130–137.

Power, R., Scott, D., & Evans, R. (1998). What You See Is What You Meant: direct knowledge editing with natural language feedback. In *Proceedings of the 13th Biennial European Conference on Artificial Intelligence* (pp. 677–681). Brighton, UK.

Procter, P. (1995). *Cambridge International Dictionary of English.* Cambridge, UK: Cambridge University Press.

Pullum, G., & Zwicky, A. (1997). *Licensing of prosodic features by syntactic rules:*

the key to auxiliary reduction. (Presented at the Annual Meeting of the Linguistic Society of America, Chicago, IL. <http://ling.ucsc.edu/~pullum/locker/lsa1997abstr.html>)

Quinlan, P. (1992). *The Oxford Psycholinguistic Database.* Oxford, UK: Oxford University Press.

Ritchie, G., Russell, G., Black, A., & Pulman, S. (1992). *Computational morphology: practical mechanisms for the English lexicon.* Cambridge, MA: MIT Press.

Sampson, G. (1995). *English for the computer.* Oxford, UK: Oxford University Press.

Shieber, S., van Noord, G., Moore, R., & Pereira, F. (1990). Semantic head-driven generation. *Computational Linguistics*, *16*(1), 7–17.

van Noord, G. (1991). *Morphology in MiMo2.* Manuscript, University of Utrecht, The Netherlands.