UPPSALA UNIVERSITY

SCARRIE

Project ref. no.	**LE3-4239**
Project title	**SCARRIE Scandinavian Proof-reading Tools**

Deliverable number	*DEL 1.2*
Deliverable title	*Testing CORRie for SCARRIE*
Number of pages	*19*
WP/Task responsible	*Anna Sågvall Hein, Department of Linguistics, Uppsala University, Box 513, S-751 20 Uppsala, Sweden.* *Email:* anna@ling.uu.se
Author	*Erik Tjong Kim Sang*
EC Project Officer	*Pierre-Paul Sondag*
Keywords	*CORRie, evaluation*
Abstract	*The Dutch spelling checking and spelling correction program CORRie has been tested to determine whether it can be used for Scandinavian languages. Fifteen different tests have been performed. They have examined four main features of the program: dictionary lookup recognition, non-dictionary lookup recognition, spelling correction and modularization.* *CORRie has passed eight of the fifteen tests. Four tests were passed partially, two tests were failed and for one test we need feedback from the user group in order to rate the performance of the program.*

Executive summary

The Dutch spelling checking and spelling correction program CORRie has been evaluated to determine whether it can be applied as a base for the software which will be developed in the Scarrie project. The evaluation scheme consisted of fifteen tests which have examined four main tasks that the program had to perform: dictionary lookup recognition, non-dictionary lookup recognition, spelling correction and modularization. CORRie has passed eight tests completely and four tests partially. The program failed on two tests. For evaluating the result of one test we need response from the user group.

CORRie performed well on spelling error detection but less well on spelling correction. The documentation of the program was incomplete. The consequence of this will be that the research partners will need extra time for getting acquainted with the software and improving the documentation. The conclusion about CORRie was the following: the software is usable as a basis for the SCARRIE project but adapting it to the level that the research partners had expected from the software will require extra resources which currently have not been taken into account in the project plan.

The tests have required modification of the CORRie program files. One of these modified files has been added to this deliverable on request of the SCARRIE partners.

Testing CORRie for SCARRIE

Erik F. Tjong Kim Sang
Department of Linguistics
University of Uppsala
erik.tjong@ling.uu.se

November 27, 1997

1. Introduction

The SCARRIE project is one of the European Union sponsored projects in the Language Engineering sector of the Telematics Applications Programme in the Fourth Framework [Com97]. It aims at developing high-quality proof-reading tools for the languages Danish, Norwegian and Swedish [Gro96]. The target user group for these tools is the Scandinavian publishing industry. Within the user group there is a need for proofreading tools which contain large word lists and are able to recognize compounds, punctuation errors and grammatical errors. Development of such proofreading tools for Scandinavian languages was hindered by the fact that the markets for these languages are small.

Rather than starting the development of the proofreading tools from scratch, the SCARRIE project will use existing technology. The suggested technological basis for the project is the Dutch spelling checking and correction program CORRie. This program was developed by Theo Vosse in his PhD project [Vos94]. It contains components like compound recognition, simple grammar checking and sound-and-spelling-based correction. Many of the CORRie components are interesting for the SCARRIE project.

CORRie is written for the language Dutch. The question is whether it is possible to adapt the program for the Scandinavian languages and if so, how much time that will take. This report addresses this question.

The report will start with a list of requirements on CORRie that have been put forward by the SCARRIE software development groups. The CORRie program will be submitted to tests to find out whether it meets the requirements or not. The tests will be grouped in sections regarding dictionary lookup recognition, non-dictionary lookup recognition, spelling correction and modularization.

The work presented in this report has been done in the framework of work package 1.1 and work package 6 of the SCARRIE project.

2. CORRie requirements

The Dutch spelling checking and spelling correction program CORRie was suggested as a basis for the software for spelling checking, spelling correction and grammar checking developed in the SCARRIE project. CORRie needs to be adapted for the Scandinavian languages and it needs to be extended with extra modules for handling tasks that are not supported by the software. There are some basic requirements that the program has to meet in order to make the adaption possible within the time schedule of the SCARRIE project. CORRie has to be able to:

Requirement 1.

handle dictionaries with more than one reading of a word,

Requirement 2.

check words against a word form dictionary and for each hit produce the category, lemma, inflectional features and frequency, preferably in the form of a tagged text,

Requirement 3.

handle dictionaries that contain 800,000 word forms,

Requirement 4.

create internal dictionaries of a size that is acceptable to the users,

Requirement 5.

present for each word that is not in the dictionary its analysis (some possible analyses: recognized compound, predefined minus word, word with a replacement suggested by correction module and unrecognized word),

Requirement 6.

recognize the difference between upper case and lower case characters,

Requirement 7.

recognize numerical expressions,

Requirement 8.

recognize compounds defined by compound rules,

Requirement 9.

recognize multi-word expressions (idioms) and words that only can appear in idioms,

Requirement 10.

handle predefined minus words by replacing them,

Requirement 11.

provide text split in sentences with sentence boundaries after headings,

Requirement 12.

generate reasonable correction alternatives with the best alternatives first,

Requirement 13.

recognize grammatical errors,

Requirement 14.

allow plugging in new recognition modules and

Requirement 15.

allow the replacement of the current correction module.

Some of the requirements have been based on the user requirements on the SCARRIE software [Gro96] [ASH97]. Apart from these project specific requirements there are two more general software requirements. First the software should run at the computer platforms of the SCARRIE developers. Second the program should be accompanied by good documentation. The results of the tests for the requirements will be described in the next four sections. The tests refer to the version of CORRie of October 29, 1997.

3. Dictionary lookup recognition

In this section we will test the SCARRIE requirements on CORRie related to dictionaries. This involves the requirements 1-5. We will start with describing what actions were necessary to get CORRie running on the machines of the Uppsala SCARRIE group.

3.1 Installing CORRie

Before we can start testing CORRie we have to install the software on our machines which are : IBM RS/6000 workstations running AIX 4.1. The CORRie package of October 29, 1997 can not be installed at our machines without making several modifications. Initial attempts to compile to software failed without any reasonable error message.

The problem which prevented a successful compilation could be solved by adding two extra compile options for the C compiler in the Makefile (suggestions by Theo Vosse and Per Weijnitz). After this modification compilation still failed but this time there was a usable error message (CORRie variable name *class* already defined in AIX library). The new problem could be solved by changing the name of a variable in the software in five files at in total eight places (suggestion by Theo Vosse). Unfortunately the change did not result in a successful compilation but in another error message (compilation of header file requested). This problem could be taken care of by adding an extra line to the Makefile (suggestion by Bart Jongejan). After this addition the software could be installed on our machines.

Modifications made to the CORRie files to able the software to be installed on IBM RS/6000 workstations running AIX 4.1: two modifications in the Makefile to handle two different problems and eight modifications in five other files (class->classt) to handle one more problem:

```
Makefile:line 1 became:CFLAGS=-g -DITIMER -Dunix -fsigned-char
Makefile:inserted after line 42:$(CC) $(CFLAGS) lrgen.c -o lrgen
compgram.main.c:line 91 became:case classt: index = nrClasses++; break;
compgram.main.c:line 496 became:if ((long int) GetInfo(node) %  ...
compgram.main.c:line 1073 became:if (TokenType(GetKey(node)) == classt ...
compoundgrammar.c:line 615 became:if ((type = TokenType(lastSymbol)) !=...
compoundgrammar.c:line 677 became:if ((type = TokenType(lastSymbol)) !=...
compoundgrammar.c:line 692 became:if ((type = TokenType(lastSymbol)) !=...
declgrammar.c:line 477 became:StoreToken(lastSymbol, classt);
makedict.c:line 282 became:case classt: index = nrClasses++; break;
compoundgrammar.h:line 5 became:classt,
```

3.2 Requirement 1

Requirement 1 states that the software should be able to handle dictionaries that contain more than one reading of a word. CORRie is able to do that. The example dictionary distributed with the software (test.idf) contains the lines:

```
aanzien    5334    N    NOUN_SING    II13
aanzien    5334    N    VERB         II3F
```

which defines the Dutch word *aanzien* as both a singular noun (esteem) and a verb (look at).

Conclusion: The CORRie dictionaries allow words to have more than one lexical category.

3.3 Requirement 2

Requirement 2 demands that the program is able to check words against a word form dictionary. Furthermore it should be able to produce with each hit the extra information stored in the dictionary (category, lemma, inflectional features and frequency) preferably in the form of a tagged text. In order to test this we have created a sample dictionary with 100,000 entries. The dictionary was created from a Swedish word list containing 1.4 million word forms extracted from the Uppsala Newspaper Corpus (fx.fre) [Dah97].

The word list contained word forms and their frequency sorted according to frequency. We have extracted the first 100,000 items of the list and converted these to the current format of the input dictionaries of CORRie. The dictionary records contain six fields: word form, frequency, style, grammatical information, replacement (for minus words) and optional information (in the example dictionary: lexical category). Since we do not have access to other information than word forms and frequency, we have filled the other fields with default information: N for style and NOUN_SING for grammatical information while the other two fields were left empty. The CORRie dictionary also contains information about lemmas: they are separated from each other by empty lines. In our test dictionary we have put every word in a different lemma.

In order for a dictionary to be usable by CORRie it has to be compiled with the program makedict. This presented no problems. The correction part of the software needs a phoneme dictionary which contains a different format than the compiled word form dictionary. This phoneme dictionary can be generated with the command makepron. Since CORRie requires the phoneme dictionary[1]. we had to run makepron in order to generate it. Makepron accepted only 95936 words of the list. The prime reason for this was that some of the characters in the words had not been specified in the default grafeme to phoneme conversion list. Makepron also reported that the block size was too small.

We have performed three action to get rid of the makepron problems. First we increased the size of the blockSize variable. Second we removed all words containing other characters than a-z, å, ä, ö and é from our word list (case-insensitive). And third we added simple grafeme to phoneme conversion rules to the file grafon.rules to take care of the four extra vowels with accents After these three changes we ran makepron again. However it still failed to recognize the words with the vowels with accents.

Makepron could not recognize the extra vowels because the dictionary lookup routines did not regard the vowels as an allowed part of words. We needed to modify the dictionary routines in order to change this. So we added the definitions of the ISO Latin 1 characters in the range 300-379 octal to the main dictionary routine. Since both makedict and makepron are dependent on the dictionary routines we had to run both another time after the change. Makedict accepted all words and makepron rejected one word without any error message[2].

The next necessary step was to generate an idiom dictionary. CORRie offers the possibility to turn off the idiom checking[3]. We chose to run the program with idiom dictionaries so we generated them with makedict and makephon. This did not cause any problems.

After this we could apply CORRie to our test text containing 7660 words. It reported a recognition failure for 10 of the approximately 50 numbers in the text. These numbers were reported as frequent words. CORRie detected 174 errors in the text, 10 proper names, 12 frequent words, 105 compounds. The majority of the words were approved of by the program and this must have been the result of the dictionary lookup.

The error report of CORRie showed a strange bug. In the text 174 words were marked as errors. 158 of these words were unique. However the error summary contained 222 unique word errors. The 64 extra words were present in the dictionary. The property that they had in common was that they started with a capital character or with a vowel with an accent[4][5].

The second part of the requirement states that CORRie should be able to show the word features it retrieved from the dictionary. It seems that the dictionary features of words are stored in the data structure *Appearance*[6]. In this data structure the fields *original* and *logfreq* should contain the data that we are looking for. We have added some code to CORRie to force the program to display the contents of these fields. We were able to retrieve the word frequency but the other parts of the data structure seem to be empty. Here is a sample of the program output:

> låt_0/0/8058/ vara_0/0/124093/ att_0/0/1654817/ han_0/0/392418/ själv_0/0/42169/ aldrig_0/0/33982/ deltog_0/0/4869/ i_0/0/2206734/ gruppen_0/0/8659/). Kotterilivet_0/0/1/ i_0/0/2206734/ Paris_2/1/7498/

åren_0/0/22067/

CORRie has attached a tag of the format _A/B/C/D to each word. A is the code for the lexical category, B show the number of lexical features, C is an approximation of the frequency and D contains extra information. The fields A and B are nearly always zero and field D is always empty. This does not correspond with our dictionary. In our dictionary all words have specified as nouns except for the word *han*:

```
...
de      614912    N    NOUN_SING
inte    545160    N    NOUN_SING
om      525049    N    NOUN_SING
ett     518980    N    NOUN_SING
han     372672    N    VERB         UIL: test
...
```

The occurrence of *han* on the first line in the example has received the same category information as the other words despite the fact that it was specified as an nonambiguous verb. Furthermore the extra information string "UIL: test" stored with the word was not present in the output of the program The only words with a non-zero category are some names like *Paris* in the example.

Conclusion: CORRie is able to check words against a dictionary. It stores the frequency of the words but it seems that it stores neither lexical category, lexical features, lemma nor the extra information that has been put in the dictionary. The error reporting part of the program displays incorrect results.

Note: In a later test we have been able to extract from the program lexical category and lexical features for Dutch data (see section 5.2). The condition for these to appear in the data structure seems to be that 1. the lexical features have to be defined, 2. the parser has to be called and 3. the words are in a sentence that could be parsed. The lexical category that we detected in this way was not the same as the lexical category in the dictionary.

Commands used for creating the input dictionary for CORRie (cut), compiling the main dictionary (makedict), creating the main phoneme dictionary (makepron), compiling the idiom dictionary (makedict), creating the phoneme dictionary for idioms (makepron):

```
cut -d" " -f1-2 /corpora/scarrie/us/fx.fre|\
  grep -v ' .*[^a-zA-ZÅÄÖÉåäöé]'|head -100000|\
  sed 's/\(.*\) \(.*\)/\2 \1/'|sed 's/$/ N NOUN_SING  /'|\
  tr ' ' '\t'|perl -e 'while (<>) { printf "$_\n"; }' > swedish.idf
makedict nldecl swedish.idf nn 2>&1|more
makepron nn grafon.rules 2>&1|more
makedict nldecl idioms.idf idiomdict
makepron idiomdict grafon.rules
corrie sample.def unt.txt -l unt.log
```

Modifications made to the CORRie files to able the software to handle a 100,000 words dictionary containing Swedish vowels and display the dictionary features:

```
makepron.c:line 28 became:static LongInt blockSize = 16384;
grafon.rules:added after line 5:# take care of Swedish vowels
                                /å/ -> o
                                /ä/ -> e
                                /ö/ -> eu
                                /é/ -> e
dictstring.c:added definitions of characters 300-379 octal to
               keyLetter, keyLetter2, letterEquivalent and
               letterEquivalent2.
Corrie2.c:line 2810 became:LSOutput(outbuf, zp->sort, wordPlusInfo,...
Corrie2.c:line 2837 became:LSOutput(zp->word, zp->sort, wordPlusInfo,...
Corrie2.c:line 2843 became:LSOutput(outbuf, zp->sort, wordPlusInfo,...
Corrie2.c:line 2858 became:LSOutput(outbuf, zp->sort, wordPlusInfo,...
```

```
Corrie2.c:added after line 2733:char wordPlusInfo[DEFAULT_BUFSIZE];
Corrie2.c:added after line 2791:sprintf(wordPlusInfo,"%s_%d/%d/%d/%s",
                      zp->word,
                      ((zp->appearance)->original)->category,
                      ((zp->appearance)->original)->number_ofFeatures,
                      ((int) pow(10.0,
                       ((double) (zp->appearance)->logfreq)/32.0)),
                      ((zp->appearance)->extraInfo)->extraInfo);
```

3.4 Requirement 3

Requirement 3 states that the program should be able of handling dictionaries that contain 800,000 word forms. We have tested this by creating a dictionary with 800,000 entries. Our dictionary came from the same source as the dictionary used in the requirement 2 test: the Uppsala Newspaper Corpus (fx.fre) [Dah97].

Again we needed to compile the dictionary with the program makedict. This program was not able to handle our dictionary. It complained about the value of an internal variable (blockSize) that was too small. We have increased this value in the code, recompiled CORRie and after this makedict was able to compile the dictionary. The program rejected one word because it was to long and it rejected 30 words without any error message. We have added 80 extra words to the input dictionary to force makedict to create a dictionary containing 800,000 words. We obtained a compiled dictionary containing 800049 words.

We also needed to run the phoneme dictionary creation program makepron agin. It succeeded in building a phoneme dictionary of 754914 words. Nearly all the missing words were rejected because they contained a characters that was not defined in the grapheme-to-phoneme rule definition file grafon.rules.

After this we applied CORRie to our test text containing 7660 words. Just like in the previous test it reported a recognition failure for 10 of the approximately 50 numbers in the text. Otherwise no problems were encountered. CORRie seems to be able to handle a dictionary of 800,000 words.

Conclusion: CORRie is able to handle dictionaries with 800,000 words.

Commands used for creating the input dictionary for CORRie (head), compiling the main dictionary (makedict), creating the main phoneme dictionary (makepron), compiling the idiom dictionary (makedict), creating the phoneme dictionary for idioms (makepron):

```
head -800080 /corpora/scarrie/us/fx.fre|cut -d" " -f1-2|\
  sed 's/\(.*\) \(.*\)/\2 \1/'|sed 's/$/ N NOUN_SING  /'|\
  tr ' ' '\t'|perl -e 'while (<>) { printf "$_\n"; }' > swedish.idf
makedict nldecl swedish.idf nn 2>&1|more
makepron nn grafon.rules 2>&1|more
corrie sample.def unt.txt -l unt.log
```

Modifications made to the CORRie files to able the software to handle a dictionary containing 800,000 words:

```
makedict.c:line 35 became:static LongInt blockSize = 16384;
```

3.5 Requirement 4

Requirement 4 demands of CORRie that it creates internal dictionaries of a limited size acceptable to the users. In order to test this requirement we have looked at the size of the dictionaries produced in the previous section for the word list containing 800,000 word forms.

CORRie created four dictionary files (nn, nn.phn, idiomdict and idiomdict.phn) and four index files for these dictionaries. The two largest files were the main dictionary nn (21.6 megabytes)[7] and the phoneme dictionary (11.6 megabytes)[8]. The other six files had a size of 26 kilobytes or less. Thus the total size of the dictionaries 33.2 megabytes for a 800,000 word dictionary. Whether this is acceptable or not is up to the users to decide. Notes which should be made here is that the information in the main dictionary is incomplete (some fields are unused) and that the grafeme to phoneme conversion rule list and the idiom list are virtually empty.

Conclusion: CORRie needed 33.2 megabutes for storing our test dictionary with 800,000 words. This size may increase because some fields in this test dictionary were left empty.

3.6 Requirement 5

Requirement 5 states that the program should be able to provide an analysis of each word that is not in the dictionary. Examples of such analyzes are recognized compound, predefined minus word, word with a replacement suggested by correction module and unrecognized word. These analyzes can be useful for the parser or other CORRie modules.

In the program we discovered an undocumented variable dumpInterneInfo which seems to be usable for forcing CORRie to show the word features. The default value of the variable is false. It can be set by adding a line to the CORRie definition file (sample.def). We changed the definition file. Now the program made a dump of most fields in the data structure *Appearance* for the words that appear in the report lists (proper names, frequent words, compounds, marked words and unusual words). Here is are some examples:

```
Ejeby 2ACEF/2/2/0/(null)/(null)/0/1/1/0/no compound/no idiom/ exists added propername/
manschetter 2ACE/3/0/0/(null)/manschetten/1000/1/1/50/no compound/no idiom/ exists added/
återströmning 28A/1/0/0/(null)/(null)/0/2/1/0/compound/no idiom/ exists/
änglamönster 3/1/0/0/(null)/(null)/0/0/1/0/no compound/no idiom//
arbetsmarknadskrisen 38/1/0/0/(null)/(null)/0/3/1/0/dubious compound/no idiom//
```

The words are followed by fourteen information fields. The fields 11, 12 and 13 contain the information that we are looking for. Here we can see that återströmning is a compound and that arbetsmarknadskrisen is a dubious compound while the other three words are no compounds. The word återströmning was present in the dictionary and the words Ejeby and manschetter were added because they were a proper name (Ejeby) or because they occurred frequently (manschetter, not specified). None of the words was part of an idiom.

Our dictionary did not contain minus word so we changed a frequent word (*han*) and a less frequent word (*egenskap*) to minus words[9]. CORRie reported an error for the less frequent word and suggested the correct replacement. The high frequency of the first word in the text prevented it from being reported as an error. Instead it was reported as an unusual word. In the error report the less frequent word *egenskap* was reported as a word with another preferred spelling. However the preferred spelling of the frequent *han* was not present in the error report:

```
egenskap 1AB/1/0/0/egenskaper/(null)/0/1/1/0/no compound/no idiom/ exists/
han 2A/13/1/0/(null)/(null)/0/1/1/0/no compound/no idiom/ exists/
```

Note: for some unknown reason CORRie took 618 seconds CPU time for processing our test text of 7660 words (almost 23 minutes real time).

Conclusion: CORRie is able to show for each word that was not present in its dictionary the analysis. All four example analyses in the requirement were detected during the test: compound, word with replacement, unrecognized word and minus word.

Commands used for obtaining the extra word analyses:

```
corrie sample.def unt.txt -l unt.log
```

Modifications made to the CORRie files for obtaining the extra word analyses:

```
Corrie2.c:added code to function DumpWb for writing out Appearance fields
Corrie2.c:added same dump code to function DumpIFW
sample.def:added after last line:dumpInternalInfo=on
```

4. Non-dictionary lookup recognition

In this section we will deal with error detection that is not based on dictionary lookup. We will test the requirements 6-11: recognizing the difference between upper case and lower case characters, recognizing numerical expressions, recognizing compounds defined by compound rules, recognizing multi-word expressions and suggesting replacements for them, handling predefined minus words by replacing them and providing text split in sentences with sentence boundaries after headings.

4.1 Requirement 6

Requirement 6 states that CORRie should be able to recognize the difference between capital characters and lower case characters. This feature will be used for correcting words that have some obligatory case in their first character. In order to test this we have applied CORRie to a text containing the words *sverige* (should be *Sverige*) and *Svenska* (should be *svenska*). Our dictionary contained the correct versions but not the incorrect ones. CORRie detected the first error but not the second.

Conclusion: CORRie can detected an error when a word that should have an initial capital character appears with an initial lower case character. It cannot detect an obligatory initial lower case character being replaced with a capital character.

4.2 Requirement 7

Requirement 7 states that the program should be able to recognize numerical expressions. Numerical expressions should be stored in a dictionary because that would take too much place. CORRie should contain a function that determines whether a string is a numerical expression or not. The results of this function should be available to the other modules in the program.

The current version of CORRie skips every string that includes a digit. According to the manual and the program code this behavior can be modified by adding an extra definition line to the CORRie definition file (sample.def)[10]. We have tried this but it did not work: after adding "digits=on" to the definition file the behavior of CORRie with respect to handling numbers remained the same. An inspection of the code revealed that the line in the definition file influences the value of a unused flag cleanUpDigits.

Conclusion: CORRie ignores every string that includes a digit. Changing this behavior requires modifications in the code[11].

Commands used for testing whether CORRie recognized numerical expressions:

```
corrie sample.def unt.txt -l unt.log
```

Modifications made to the CORRie files for making CORRie accept numbers in words:

```
sample.def:added after last line:digits=on
```

4.3 Requirement 8

Requirement 8 states that CORRie should be able to recognize compound by using a list of compound rules. The CORRie compound rules are stored the file nlregexp[12]. This distribution of CORRie was shipped with two compound rules for Dutch. In section 3.6 we have seen two examples of reasonable compounds which were formed with these rules.

Conclusion: CORRie is able to recognize compounds which are specified in a list of compound rules.

4.4 Requirement 9

Requirement 9 states that the program should be able to recognize multiword expressions and words that only can

appear in these expressions. Multiword expressions (idioms) are defined in the file idioms.txt. As a test we added the Swedish idiom *huller om buller* to this file. Furthermore we defined *huller* in our lexicon as a word that is only allowed in idiomatic expression. After this we tested the program with the correct idiom and three misspellings. The result was the following:

```
    Huller om buller.
    #1#Huller om bullar.
--> 1.Heller
    Huller om #2#bullir.
--> 2.buller
    #3#Hullar om buller.
--> 3.Heller
    Upsala Nya Tidning är den största lokala tidning i #4#Upsala
--> 4.Uppsala
```

The first phrase was correct. In the second phrase we had replaced *buller* by the existing word *buller* and the program flagged *Huller* as an error because it appeared outside the idiom context. In the second phrase *buller* was replaced with the non-existent *bullir*. That word was recognized as an error and *Huller* was accepted. The third phrase contained the non-existent *Hullar* instead of *Huller*. The error was recognized but the suggested correction did not use the idiom context. And in the final phrase there are two occurrences of the old spelling *Upsala*. The first one is correct because it occurs in a predefined idiom but the second is wrong.

Conclusion: CORRie is able to recognize idioms and report idiomatic words that appear outside of an obligatory idiom context. The error correction phase does not seem to make use of idiom information. Idiom handling in CORRie is undocumented.

Commands used for testing processing of multiword phrases:

```
makedict nldecl swedish.idf nn
makedict nldecl idioms.idf idiomdict
makepron idiomdict grafon.rules
corrie sample.def tmp.txt -l -
```

Modifications made to the CORRie files for testing processing of multiword phrases:

```
idioms.txt:added after last line:huller om buller : ADV([enigma])
                                 Upsala Nya Tidning : NK(enigma sg3 _ _)
swedish.idf:defined huller and Upsala as idiom words
```

4.5 Requirement 10

Requirement 10 states that CORRie should be able to handle predefined words by replacing them. Words in the lexicon can be defined as minus words by putting an R in the third field of their lexicon entry and placing a replacement word in the fifth field. We have defined the word *huller* as a minus word with *buller* as its replacement. The program recognized the word as an error and suggested the correct replacement[13]. It did not make the replacement automatically.

We were also interested in covering multi-word spelling errors like *i genom* which should have been *igenom*. We have not been able to make CORRie recognize this error. The phrase *i genom* can be added to the dictionary as a minus phrase but the space in the word will be ignored during processing. The idiom word list does not seem to allow negative phrases.

Conclusion: CORRie is able to recognize predefined minus words and suggest the correct replacement but it does not make the replacement itself. It seems it cannot handle minus phrases that are longer than one word.

Commands used for testing minus words:

```
makedict nldecl swedish.idf nn
makedict nldecl idioms.idf idiomdict
makepron idiomdict grafon.rules
corrie sample.def tmp.txt -l -
```

Modifications made to the CORRie files for testing minus words:

```
idioms.txt:remove line:huller om buller : ADV([enigma])
swedish.idf:defined huller as a minus word
```

4.6 Requirement 11

Requirement 11 states that the software should be able to provide the text divided in sentences with sentence boundaries behind headings. For the recognition of headings we have assumed that they are followed by two line breaks without any non-white space characters between the line breaks and the final character of the heading. It is possible in CORRie to define a paragraph boundary string with the routine LSDefParagraph. We have tried this but observed that the last sentence before the paragraph break disappeared from the output logs.

We manged to make the program produce output text with sentence boundary strings by adding some code to the routine ProcessSentence. The output of the program was like the following:

> Männen hittar kvinnan. <s>Efter förberedelser av sina nya utrikesministerar. <s>

Conclusion: CORRie is able to provide text divided in sentences. Getting headings tagged as sentences failed because of a possible programming bug.

Commands used for obtaining sentence delimiters in the output of the program:

```
corrie sample.def sparse.txt -l -
```

Modifications made to the CORRie files for obtaining sentence delimiters in the output of the program:

```
Corrie2.c:added before final line in ProcessSentence:
   if (sentenceText != NULL) {
      LSOutput(outbuf, LSPunctuation, "<s>", NULL, false);
      StringLog(LSPunctuation,outbuf);
   }
```

5. Spelling correction

This section describes the tests we have performed with the spelling correction facilities in CORRie. We have tested two requirements: number 12 which requires that CORRie generates reasonable correction alternatives and requirement 13 which states that the program should be able to recognize grammatical errors.

5.1 Requirement 12

Requirement 12 states that the program should be able to generate reasonable alternatives for incorrect words with the best alternatives first. The present version of CORRie returns zero or one alternative for each incorrect word. The proposed alternatives are influenced or perhaps even based on a phonetic dictionary build by the program makepron which should replace the trigram and triphone programs which huge generated dictionary files. This program makepron is undocumented.

In order to test the correction alternatives from CORRie we have applied the program with a 100,000 word dictionary to our 7660-word test text from Upsala Nya Tidning. The program reported 193 errors and we have performed a manual check on the first 100 errors (see appendix A for a list of these 100 errors). In twelve cases the program suggested the concatenation of words in phrases that contained two or three words. All suggestions were wrong. Most words in the suggested concatenations were in the lexicon while the concatenations themselves were not.

For 49 of the remaining words CORRie did not suggest any alternative. The alternatives suggested for the other 39 words seem reasonable most of the time. Strange cases like the suggestion of *frågade* for *frossade* can be explained by the fact that the suggestion is the most common alternative. More and better correction alternatives can probably be obtained by increasing the dictionary and grouping related words in lemmas[14]. According to [Vos97] it should also be possible to make makepron generate more alternatives but this will result is a larger phoneme dictionary file.

Conclusion: With a 100,000-word lexicon CORRie does not generate any alternative for approximately 50% of the words that are reported as errors. It generates maximally one alternative per reported error. The alternatives suggested for single word errors are reasonable. Neither triphone nor trigram analysis is available as a default. The program makepron is important for the correction process but it is undocumented.

Commands used for testing the correction module:

```
makepron nn grafon.rules 2>/dev/null
corrie sample.def unt.txt -l unt.log
```

5.2 Requirement 13

Requirement 13 states that CORRie should be able to recognize grammatical errors. This means that the program should be able to parse at least phrases. This version CORRie of contains a build-in parser. The parser is normally not used by the program. It can be included in the processing stage by using an extra option while starting CORRie. The grammar rules can be found in the file smpgram.

We have tested the parser by using the Dutch parsing rules that came with the program. We have replaced the Dutch parsing test sentences with Swedish sentences and have added the required lexical information to our Swedish dictionary of 100,000 words. We made up different versions of a Swedish verb in order to be able to test the same grammatical problem as in the Dutch test file that was supplied with the program. CORRie recognized the two errors it should have recognized. It generated two kinds of output. First it generated tags for each word in the sentence in the standard log file:

Kvinnan_/AP/0/9305/ hittar_/V/1/3398/ mannen_/AP/0/20535/.

The verb *hittar* has been tagged as *V* and the two definite nouns *kvinnan* and *mannen* have been tagged as *AP* (adjective phrase). Apart from this the program stored a parse of each sentence in a file tree.log. The parses look something like this:

```
(S
   (NP([sg1 sg2 sg3 pl] [acc nom dat])
      (ArtOpt)
      (APOpt)
      (N([sg1 sg2 sg3 pl]) Kvinnan))
   (VP([sg1 sg2 sg3 pl])
      (V([sg1 sg2 sg3 pl]) hittar)
      (NP([sg1 sg2 sg3 pl] [acc nom dat])
         (ArtOpt)
         (APOpt)
         (N([sg1 sg2 sg3 pl]) mannen)))
   (Punc .))
```

The different tokens in this parse tree are explained in the manual and listed in the grammar files smpgram and smpgram.lex. It is strange that all words have been tagged as the singular version of three person forms. We have been able to remove the extra analyses for the object word by adding a line to the lexical grammar file smpgram.lex. This approach did not succeed in removing the extra analyses for the subject word.

It is good to take a look at the dictionary files. The input dictionary looks like this:

```
hitta      9281    N    VERB         II3C
hittar     3314    N    VERB         3E3C
hitta      3314    N    VERB         AM3C
hittade    2621    N    VERB         VE3C
hittade    2621    N    VERB         VM3C
man      247147    N    NOUN_SING    II11
mannen    20500    N    NOUN_SING    3E11
männen     6655    N    NOUN_PL      MV11
```

Contrary to what we expected it is the final field that determines the lexical category, not the fourth field. The four-character codes in the final field are specified in the lexical grammar file smpgram.lex as follows:

```
II3C    V(inf)
3E3C    V(sg3)
II11    N(sg3)
MV11    N(pl)
```

We will not be able to parse the complete input of the program. Therefore we want to use the parser with so-called local error rules: rules that parse phrases instead of complete sentences. We wanted to know whether the CORRie parser was able to do this. Therefore we have adapted the call to the parser from CORRie's main routine and forced the program to call the parser for each word bigram and each word trigram[15]. Then we added two local error rules to the grammar of the program and adapted the lexical rules and the dictionary in such a way that the program was able to handle the two test sentences that are part of the Uppsala Error Database for Swedish. The output of the program for our four input sentences was as follows:

```
    Efter förberedelser av sina nya utrikesministrar ...
    Efter förberedelser av #1#sina nya utrikesminister ...
--> 1.sin
    Om människor kan börja ...
    Om människor #2#börja tro ...
--> 2.börjar
```

CORRie correctly recognized the agreement errors in the second sentence (sina=plural, utrikesminister=singular) and the one in the fourth sentence (människor=plural, börja=infinitive). It also suggested the correct replacements. The first and the third sentence are correct. The local error rules that we have added to the grammar are:

```
S --> Pro(PersNumber:2 gen) Adj(def) N(PersNumber:10)
S --> V([sg1 sg2 sg3 pl]) V(inf)
```

The first rule states that in a sequence of a genitive pronoun, an adjective and a noun the following should be true: the adjective should be definite and the number of the pronoun and the noun have to be the same. The numbers 2 and 10 were necessary to make the parser report an error for the pronoun rather than for the noun. The second rule states that in a verb bigram the second verb must be an infinite verb while the first one cannot be an infinite verb[16].

Conclusion: CORRie is able to recognize grammatical errors. It can recognize certain grammatical errors without requiring complete parses of sentences.

Commands used for abling CORRie to parse its input: these are the two standard commands for creating the dictionary plus an extra command for compiling the parse rules (lrgen). Running CORRie with parsing turned on requires an extra option (+samplep.def):

```
makedict nldecl swedish.idf nn
makepron nn grafon.rules
lrgen smpgram smpgram.tbl
corrie sample.def +samplep.def sparse.txt -l -
```

Modifications made to the CORRie files for testing the parser:

```
Corrie2.c:modifications around the CheckSentence call to force the
          program to call this function for all bigrams and trigrams
smpgram:added DefType = def odef and changed all AP and Adj
         accordingly
smpgram:added local error rules:S -> V([sg1 sg2 sg3 pl]) V(inf)
           S -> Pro(PersNumber:2 gen) Adj(def) N(PersNumber:10)
smpgram.lex:line 38 became:BF21    Adj(def)
smpgram.lex:all other Adj became Adj(odef)
smpgram.lex:line 176 became:II3C    V(inf)
smpgram.lex:added after last line:3E11    N(sg3)
                                  3E12    N(sg3)
                                  PS11  Pro([sg1 sg2 sg3] gen)
                                  PP11  Pro(pl gen)
swedish.idf:created lemmas for borja, ny, sin, tro and
            utrikesminister
```

6. Modularization

This section contains a description of tests which deal with modularity issues in CORRie. First we will try to add a new error recognition module to the program (requirement 14) and then we will attempt to replace the correction module (requirement 15).

6.1 Requirement 14

Requirement 14 states that it should be possible to add new recognition modules to the program. These new recognition modules can take care of recognizing special word and special phrases, both correct and incorrect variants. As a test we have tried to add a module that recognized negative idioms. In order to not complicating things we have restricted the module to one specific idiom: *i genom* which should be spelled as *igenom*.

We have made a modification to the routine ProcessSentenceElement in the file Corrie2.c. When the routine observes the sequence *i genom* it will mark the words as errors with replacement word *igenom* by changing values in the appearance fields of the words. This approach failed. We observed two behavior patterns of the program. It is possible to declare the words as nonexistent and then the words in the phrase *i genom* will be marked as errors with the correct replacement. However a side-effect of this is that all single occurrences of *i* and *genom* will be marked as errors. An alternative solution would be to mark the words as idiomatic errors. When we tried this CORRie failed to recognize the words as errors probably because the idiomatic error flag is only checked for idiomatic words.

Conclusion: Adding new error correction modules to CORRie will require in some cases a modification of existing parts of the software that in principle should not have anything to do with the new modules.

Commands used for testing the addition of a new error recognition module:

```
corrie sample.def tmp.txt -l -
```

Modifications made to the CORRie files testing the addition of a new error recognition module:

```
Corrie2.c:modifications in the ProcessSentenceElement function
          for recognizing the phrase "i genom" as an error and
          changing the appearance fields of the words for marking
          them as errors
```

6.2 Requirement 15

Requirement 15 states that it should be possible to exchange the correction module of CORRie with some other correction module[17]. The original modules of the program that worked with trigrams and triphones are an important part of the SCARRIE project. These modules are no longer present in the software because they generated large dictionary files. They have been replaced with the undocumented program makepron.

CORRie has a possibility for compiling without the correction module by the addition of an extra compiler option to the Makefile. We have attempted compilation in this way. After the compilation CORRie did not generate any alternatives for words that were recognized as errors[18]. The required time for processing out test file went down with 60%. We attempted to implement an alternative correction module by adding extra code to the function ProcessSentenceElement in the file Corrie2.c. The code made modifications in the appearance data structure for nondictionary words in such a way that the word *perfekt* became the best alternative for them. This worked.

Conclusion: It is possible to replace the CORRie correction module with another one. The current correction module makepron can be disactivated by using the extra __NOCORRECTION__ compile directive. Some extra actions are necessary to remove the dependency on the phoneme dictionary files.

Commands used for testing the replacement of the correction module:

```
corrie sample.def tmp.txt -l -
```

Modifications made to the CORRie files for testing the replacement of the correction module:

```
makefile:line 1: added -D __NOCORRECTION__
Corrie2.c:code additions to the ProcessSentenceElement function
          for forcing the word "perfekt" as a good alternative
          for every misspelled word.
```

7. Concluding remarks

We have tested the spelling correction program CORRie against fifteen requirements. The program has passed eight requirements (1, 3, 5, 8, 9, 10, 13 and 15) and failed two (7 and 14). Four requirement were passed partially (2, 6, 11 and 12) and for one requirement we need response from the user group (4). CORRie seems to perform reasonably well for spelling error detection but less well for spelling correction. We were happy about the parser working well and the successful test with the replacement of the correction module. However we are worried about the failure of the addition of an extra error recognition module because some extra recognition modules are called for in the SCARRIE project.

The manual pages of the software distribution lack features we expect from good documentation: good structure, completeness and correctness. We have been unable to find any documentation on new makepron program. We also noticed the absence of comment lines in code at places where they would have been very helpful. However nearly all tests were conducted without having to consult the CORRie programmer so by investing time in reading both the code and the manual pages it is possible to obtain a basic idea of how the program works. The present state of the manual pages implies that the research partners will have to invest time for writing and rewriting them. As far as we know no resources have been invested for this task in the SCARRIE project.

Our final conclusion about CORRie is the following: the software is usable as a basis for the SCARRIE project but adapting it to the level that the research partners had expected from the software will require extra resources which currently have not been taken into account in the project plan.

8. Remaining work

Based on the results of our tests we would like to suggest to start working on the items in the following list for adding extra functionality to CORRie:

1. recognition and handling minus phrases of two words and longer (for example *i genom* which should have been *igenom*),

2. recognition and handling words that have been split because of extra spaces (for example *fastslo g* which should have been *fastslog*),

3. recognition and handling numerical expressions (for example *80* which might occur in *80-talet*),

4. handling concatenated words (illegal compounds),

5. recognition of word that incorrectly start with an initial capital character (like *Svenskarna* which should have been *svenskarna* and

6. writing better documentation.

We estimate that items 1-5 will take at least a person week each while item 6 might take between four and six person weeks provided that it is done by someone that understand the program.

Notes

1. The __NOCORRECTION__ compiling definition does not work, see section 6.2.

2. The latter problem was already reported to Theo Vosse on June 14, 1997 but it is still present in the October version of the program.

3. Idiom checking can be turned off by removing the line dictionary=idiomdict from the file sample.def.

4. This problem was already reported to Theo Vosse on June 14, 1997 but it is still present in the October version of the program.

5. There were three extra words that neither started with a capital character nor included a vowel with an accent. Two of them contained a hyphen and the third was a compound.

6. The documentation about the data structure *Appearance* is inconsistent with the software.

7. The size of the input dictionary was 23.4 megabytes.

8. Approximately 3% of the words had rejected by makepron because they contained characters which were not allowed in words, for example digits.

9. The dictionary frequency of minus words must be 0 otherwise CORRie will ignore the minus word flag and treat the word as a normal word.

10. The manual and the code are inconsistent on this point.

11. It has to do with the IsWord2 function in the file Corrie2.c.

12. The name of the compound rules file is not obvious and it is hard to find the name in the manual since it is mentioned only once in a file which is supposed to be out of date.

13. The recognition of the the negative words did not work when *huller* was defined as a part of an idiom while at the same time the frequency of the word defined in the dictionary was larger that zero.

14. In our test dictionary every word was put in a separate lemma.

15. Our modified version of CORRie parses every bigram and every trigram. In non-toy applications this might proof to take too much time.}.

16. We are aware of the fact that there are exceptions to this rule.

17. Originally we required the parsing module to be replaceable as well but since this module performs reasonable replacement might not be necessary.

18. Some phoneme dictionary has to be present otherwise the program will crash.

References

[ASH97]

Anna Sågvall Hein. *Input to the Scarrie grammar checking module*. BSCW working space: http://bscw.gmd.de/bscw/bscw.cgi/d1068294/wp6a.html, 1997.

[Com97]

European Commission. *Language Engineering: Harnessing the Power of Language*. http://www2.echo.lu/langengen/lehome.html, 1997.

[Dah97]

Bengt Dahlqvist. *Word Frequency Lists for the Uppsala Newspaper Corpus*. Collection of Swedish word lists from the 67 million word Uppsala Newspaper Corpus, 1997.

[Gro96]

SCARRIE Group. *SCARRIE Scandinavian Proof Reading Tool*. http://www2.echo.lu/langeng/en/le3/scarrie/scarrie.html, 1996.

[Mus97]

Bradley Music. *Replacing the CORRie Parser*. http://bscw.gmd.de/bscw/bscw.cgi/d2759161/RepParser.html, 1997.

[Vos94]

Theo Vosse. *The Word Connection*. Enschede Uitgeverij, The Netherlands, 1994. ISBN 90-75296-01-0.

[Vos97]

Theo Vosse. *A Functional Description of CORRie*. http://ruls41.fsw.leidenuniv.nl/cordoc/, 1997.

Appendix A: Reported errors with correction suggestions

This is a list of 100 reported errors reported by CORRie with a 100,000 word Swedish dictionary for a test input file from Upsala Nya Tidning with 7660 words. This error report was generated while testing requirement 12 (section 5.1). The list contains an error number, a recognized misspelling and a suggested replacement (sometimes none),

1. Nyskaparen (Nyskapare)
2. Claes af Geijerstam (Concatenate "ClaesafGeijerstam")
3. ERIK SATIE (Concatenate "ERIKSATIE")
4. Socrate (Sockrat)
5. nyskaparen (nyskapare)
6. Erik Satie (Concatenate "ErikSatie")
7. Satie som människa (Concatenate "Satiesommänniska")
8. företal (företag)
9. PARIS VAR Europas (Concatenate "PARISVAREuropas")
10. häxkittel
11. kotterierna
12. tonsättargruppen
13. formerades (formerade)
14. smockorna
15. dadaismen
16. brännpunkten (brännpunkt)
17. Sarcueil-Cachan
18. erbarmliga (erbarmligt)
19. livas (levas)
20. Kotterili
21. CLAES AF GEIJERSTAM (Concatenate "CLAESAFGEIJERSTAM")
22. tygmässor
23. klädmode
24. syntetgarn
25. nykonstruerade
26. handknypplade
27. Duhs (Des)
28. spetsarna (spetsar)

29. Spetsarna (Spetsar)
30. hippiechict
31. frossade (frågade)
32. klädedräkten (klädedräkt)
33. hålsöm
34. vitbroderi
35. knypplade (knappade)
36. flätning (flyttning)
37. snörmakeri
38. silvertråd
39. Estravagansen
40. ståndens (ståden)
41. flamländsk (flamländska)
42. spetsbårder
43. Brysselspetsar
44. svartknypplad
45. Chantillyspets
46. boleror (bolero)
47. knypplad
48. virkad (verkar)
49. infällda (ifyllda)
50. Meulen
51. CHRISTINA SEDWALL (Concatenate "CHRISTINASEDWALL")
52. Ting-Huset i Sundsvall (Concatenate "Ting-HusetiSundsvall")
53. dopklänningar
54. busigt (busiga)
55. rumspalmer
56. mässarrangemang
57. keruber (kurirer)
58. änglamönster
59. tebutik

60. boutique
61. Pia Högström (Concatenate "PiaHögström")
62. Högströms (Högström)
63. Pia Högström (Concatenate "PiaHögström")
64. Gölin (Göken)
65. vägledningar (vägledning)
66. betats (betalt)
67. Vadoran
68. Brädväggarna
69. konstnärsateljen
70. vindskontor
71. ostädat (oskyddat)
72. tonårsrum
73. medaktörerna
74. tonårings (tonåring)
75. medverkandes (medverkande)
76. Saara (Sara)
77. Röor (Rör)
78. Camilla Wittmoss (Concatenate "CamillaWittmoss")
79. pjästexten
80. sketcherna (sketcher)
81. vidhäftade
82. lockropen (lockrop)
83. solariemarknaden
84. självbilder (självbilden)
85. jagkänsla
86. måttlösa
87. yvighet (evighet)
88. svärmerier (svärmeri)
89. Onani
90. genomskärning

91. Fib (Feb)
92. Leslie-Spinks
93. BO-INGVAR KOLLBERG (Concatenate "BO-INGVARKOLLBERG")
94. gemaket
95. urträde
96. skaderisker (skaderisken)
97. personalvård
98. beskärningarna (beskyllningarna)
99. skattekostnad
100. övertrassering

References

[ASH97]

Anna Sågvall Hein. *Input to the Scarrie grammar checking module*. BSCW working space: http://bscw.gmd.de/bscw/bscw.cgi/d1068294/wp6a.html, 1997.

[Com97]

European Commission. *Language Engineering: Harnessing the Power of Language*. http://www2.echo.lu/langengen/lehome.html, 1997.

[Dah97]

Bengt Dahlqvist. *Word Frequency Lists for the Uppsala Newspaper Corpus*. Collection of Swedish word lists from the 67 million word Uppsala Newspaper Corpus, 1997.

[Gro96]

SCARRIE Group. *SCARRIE Scandinavian Proof Reading Tool*. http://www2.echo.lu/langeng/en/le3/scarrie/scarrie.html, 1996.

[Mus97]

Bradley Music. *Replacing the CORRie Parser*. http://bscw.gmd.de/bscw/bscw.cgi/d2759161/RepParser.html, 1997.

[Vos94]

Theo Vosse. *The Word Connection*. Enschede Uitgeverij, The Netherlands, 1994. ISBN 90-75296-01-0.

[Vos97]

Theo Vosse. *A Functional Description of CORRie*. http://ruls41.fsw.leidenuniv.nl/cordoc/, 1997.