Electronic text corpora are collections of texts that represent a language. Various kinds of corpora can be distinguished. Perhaps the most common type of text corpus is a collection of texts that tries to include all aspects of language use and then represent the average use of the language. There are two basic ways of compiling such general corpora. In one approach, all different text types are represented in the corpus in the same proportion regardless of how common the text type is in use. In another approach, a corpus is compiled so that each type of text, or each genre of text, is represented in the corpus in the proportion that it has in language use in general. Thus common news would have a larger part in the corpus than astronomy, for example. Such collections of texts are called balanced corpora. The latter type of corpus is certainly more representative of the whole language use than the former type.

In addition to general purpose corpora, also specialised corpora can be compiled. Such corpora have typically texts from one domain or sub-area of language use. For example, the corpus may have only texts related to jurisprudence, or administration, or linguistics, etc. Corpora containing texts of only one sub-field are usually intended to be some kind of store-houses of language use in that particular field. They contain lexical information of that field. More specifically, they are store-houses of domain-specific terms, where the terms of the field are stored in natural contexts. With an appropriate extraction system, such domain-specific terms can be retrieved from the corpus for such purposes as domain-specific dictionaries, for example. Such corpora serve also as domain-specific archives where the use of terms can be found in natural contexts.

We can make two rough dichotomical divisions within the field of corpus linguistics. First, the corpora can be divided into general purpose and special purpose corpora, as we have shown above. The second dichotomical division concerns the methods of handling corpora. As we very well know, there is not much point to create electronic corpora if we do not have efficient and reliable tools for information extraction. There are two very different approaches for ensuring that information extraction will be accurate. In one method the text is tagged, i.e. provided with (usually) linguistic codes for helping the search. There are several tagging schemes, depending on the purpose of the corpus, and on the resources available. Traditionally
corpus tagging has been very labour-intensive, especially if the size of the corpus was in the region of one million words, which was considered very big less than 20 years ago. The level of tagging in those corpora was the marking of the word category of each word-form, hence part-of-speech tagging, in brief POS tagging.

Because tagging had to be done manually before the development of automatic taggers, the level of tagging usually stopped there. Syntactic coding, for example, was absent, as well as semantic information. The possibilities for tagging large texts have increased dramatically when new and sophisticated programs have been developed. With the help of these new taggers it is possible to tag texts automatically on several levels, depending on need. In addition to part-of-speech, it is possible to give lemmas, full morphological information, syntactic codes, semantic codes, etymology of words, etc. The huge advantage of the automatic taggers can be seen when we compare the work and time needed for manual tagging and automatic coding. In the COBUILD project, a new corpus of 200 million words of British National Corpus (BNC) was compiled, and it was encoded with a tagger designed for tagging written standard Enflish text. The work was done by a PhD student as an additional job along with his main work.

There are two ways of working with such tagged corpora. In one method the corpus is first tagged by using a tagging program and this tagged corpus is stored as a file, to be used as source of information extraction. In such a tagged corpus all tags are visible, and queries can be formulated on the basis of what the user sees appropriate when examining the corpus.

There is another method that does not require a permanent tagged corpus at all. The encoding of text as well as subsequent information extraction tasks are performed as one single operation. It means that in conjunction with each search operation the text is first encoded, although this encoded corpus does not become visible, because subsequent search operations destroy it and leave only the search results left.

In principle both of the above methods are identical, because the level and type of coding in both methods is the same. Which method is used depends on such factors as: (1) the size of the original untagged corpus, because tagging increases its size multifold; (2) the experience of the user of the corpus (because tags do not become visible, the user has to know which they are); (3) whether the user wants to check and possibly correct the tagging result, etc.

Can we rely on the accuracy of the tagging tools? The answer is that there are bad, intermediate and good, even very good, taggers of English. For other languages the choice is much more limited. The result of bad taggers is so poor that it has to be manually corrected before use. The accuracy of the best taggers is so good that an experienced person tagging texts manually makes more mistakes than the good tagging program.

1. Tools for information extraction

There is a large number of possibilities for carrying out information extraction and several tools are available for doing the task. Here we shall make a brief review of three main types of such
tools, i.e. (1) tools operating on the basis of concrete string matching, (2) tools which operate on the basis of string matching but which also utilise regular expressions, and (3) tools which perform linguistic analysis of text and make it possible to utilise tags in search.

There are tools that operate in several operating systems, and others are restricted to one system only. In order to bring the discussion to an operational level, in the following it is assumed that we work in the Unix/Linux environment. We also utilise the tools that come as part of those systems, such as the grep family of information extraction tools, and the piping mechanism. We shall also use some tools that are not part of the general package. These tools can be divided into two groups: (1) extended grep tools developed for corpus research, and (2) language-specific tools based on detailed language analysis. Demonstration will be done with the Swahili Language Manager (SALAMA), developed in Helsinki within the course of the past 15 years. SALAMA includes a host of tools, the most important ones being SWATWOL (Two-Level Analyser of Swahili) and SWACGP (Constraint Grammar Parser of Swahili).

1.1 Tools based on string matching

These are fairly simple tools, which can be used for extracting strings, normally words, from text. One may retrieve the required words only, but usually one wants some context also to see in which context the word is used. One may also extract collocations, or even sequences of several words, all with context. The amount of context may vary. Often the sentence length is enough. Sometimes more context is needed. For quick look one might prefer a format where the key word is always located in the same position of the line and the findings are arranged in an alphabetical order according to the keyword and its right context. Context is restricted to what fits on the line, but this is often enough for such a purpose.

It is sometimes useful to have more context than just the line or the sentence where the hit was found. The user may be given a possibility to define the number of previous and/or subsequent lines as context. Likewise it is often necessary to keep track on where the hit was found in the document. The corpus may consist of a large amount of separate texts, each marked for its name, date of production, page numbers in original print, etc. All or part of this information may be given in conjunction with the hit and its context.

Tools based on string matching are very popular, because they are simple to use. One needs only to invite the program and type the string, often called pattern, and perform the search task from the corpus. For many purposes this is sufficient and no more sophisticated programs are needed. For languages with little morphology, such as English, a string search tool is ideal, if no linguistic information is searched.

On the other hand, for languages with rich morphology simple string search is often not sufficient. Think about verb morphology in Bantu languages, where both inflection and derivation processes make the verb structure very complicated. The noun class structure with concordial agreement in verb prefixes complicate the structure even further. Therefore, in order
to make string search meaningful it is necessary to generalise the search string by allowing the use of sets as search string, or as part of it. Here we come to the next type of search tools, which utilise regular expressions.

We take a couple of examples to demonstrate how simple string matching works and what are its limitations.

**Task 1. Extract the words that start with 'mwana'.**

We may formulate the query:

```bash
tuuri$ cat hadithi.snt | egrep ' mwana'
```

All such words will be extracted that start with mwana. Also the word 'mwana' itself will be retrieved. What happens if the word starts with an upper case letter?

This was a simple task. Let us now retrieve a set of words that have common distinctive features in the beginning and in the end of the word.

**Task 2. Extract deverbal nouns that end in -aji.**

We know that usually those nouns belong to classes 1/2 and 11, the class prefixes having the form m-, mw-, wa-, and u-. Now we know the possible beginning and end of the required words, but we have no way to show what should be there in between. The task has to be carried out in two phases, so that first we extract words according to one feature or set of features and then give the result of this process to another program that extracts words according to another feature. We also have to note that we cannot retrieve context to the hit, because it may turn out that the second run finds words that were context in the first run. Because in the simple string search we do not have alternation in use, we have to run the program according to each criteria separately. Thus:

```bash
tuuri$ cat hadithi.snt | sanalista | egrep ' m' > res1
tuuri$ cat hadithi.snt | sanalista | egrep ' w' > res2
tuuri$ cat hadithi.snt | sanalista | egrep ' u' > res3
```

Then we run:

```bash
tuuri$ cat res1 res2 res3 | egrep 'aji ' > aji.res
```

All this is complicated and requires the creation of three temporary files. Yet the result is not
accurate at all. Below we shall perform the same task in two other methods.

1.2 Tools based on string matching and regular expressions

Here is a brief description of what the concept 'regular expression' means. We shall use them below in formulating query strings.

REGULAR EXPRESSIONS

A 'regular expression', or regexp, is a way of describing classes of strings. The simplest regular expression describes itself. For example 'abc' matches the string abc.

The regular expressions get more complicated when special characters are added. The special characters of regular expressions:

^ matches the beginning of the string or the beginning of a line within the string. For example:
^Chapter matches the string Chapter at the beginning of a string. This character can be used to identify the beginning of a line. (NOTE that in 'Unix world' everything that is between two hard returns is considered as one line, even though the line continues to the next 'line' on the screen. This is called 'a long line' and at the end of each 'line' on screen there is a backslash to tell the user that the line continues.)
$
\$ is similar to '^', but it matches only at the end of a string or the end of a line within the string. For example
o$
matches a string that ends with a vowel 'o'.
. matches any single character except a newline. For example
.P matches any single character followed by a 'P' in a string. Using concatenation you can make regular expressions like
U.A which matches any sequence of three characters that begins with 'U' and ends with 'A', for example 'USA'
[...] Square brackets form a 'character set'. It matches any of the characters enclosed in the square brackets. For example [MVX]
matches any of the characters 'M', 'V', or 'X' in a string. Ranges of characters are indicated by using a hyphen between the beginning and ending characters, and enclosing the whole thing in square brackets. For example

[0-9]
matches any digit.

[^[...]] This is a 'complemented character set'. The first character after the '[' must be a '^'. It matches any characters except those in the square brackets. For example

[^0-9]
matches any character that is not a digit.

| This is the 'alternation operator' and it is used to specify alternatives. For example

[^P][0-9]
matches any string that matches either '^P' or '[0-9]', i.e. any string that starts with 'P' or contains a digit. The alternation applies to the largest possible regular expression on either side.

(...) Parentheses are used in regular expressions for grouping. They can be used to concatenate regular expressions containing the alternation operator '|'. For example

possibilit(ies)
matches either 'possibility' or 'possibilities'.

* means that the preceding regular expression is to be repeated 0-n times. For example

ph*
applies the '*' symbol to the preceding 'h' and looks for matches to one 'p' followed by any number of 'h's (including zero). The '*' repeats the 'smallest' possible preceding expression. If you need to repeat a larger expression, use parentheses. For example

(abc)*
matches '', 'abc', 'abcabc', 'abcabcabc', etc.

+ is similar to '*', but the preceding expression must be matched at least once. This means that

wh+y
would match 'why' and 'whhy' but not 'wy', whereas 'wh*y' would match all three of these strings.

? is similar to '*', but the preceding expression can be matched once or not at all. For example

fe?d
will match 'fed' or 'fd', but nothing else.

\ is a backslash. It has two functions: it is used to suppress the special meaning of a character (including '\' itself), and it introduces additional special constructs (see below). For example

\$
matches the character '$'.

To match '-', write it as '---', which is a range containing only '-'. You may also give '-' as the first or last character in the set. To match '^', put it anywhere except as the first character of a set. To match a ']', make it the first character in the set. For example

...
[]d^]
matches either ']', 'd' or '^^'.
Usually '\' followed by any character matches only that character. However, there are several
exceptions: some characters form special regular expression constructs when preceded by '\'.
|            specifies an alternative. Two regular expressions a and b with '|' in between form an
expression that matches anything that either a or b will match. Thus, 'uta\|ona' matches either 'uta'
or 'ona' but no other string. '|' applies to the largest possible surrounding expressions. Only a
surrounding '( ... )' grouping can limit the grouping power of '|'. Full backtracking capability
exists to handle multiple uses of '|'.
( ... )   is a grouping construct that serves three purposes:
1. To enclose a set of '|' alternatives for other operations. Thus, '(uta\|ona)x' matches either 'utax'
of 'onax'.
2. To enclose a complicated expression for the postfix '*' to operate on. Thus, 'ba\(na\)*' matches
'bannana', etc. with any (zero or more) number of 'na' strings.
3. To mark a matched substring for future reference. This last application is not a consequence of
the idea of a parenthetical grouping; it is a separate feature which happens to be assigned as a
second meaning to the same '( ... )' construct because there is no conflict in practice between the
two meanings.
\ matches an empty string, provided it is at the beginning of the buffer.
\ matches an empty string, provided it is at the end of the buffer.
\b matches an empty string, provided it is at the beginning or end of a word. Thus, '\butu\b'
matches any occurrence of 'utu' as a separate word. '\bbahari?\b' matches 'bahari' or 'bahar' as a
separate word.
\B matches an empty string, provided it is NOT at the beginning or end of a word.
< matches an empty string, provided it is at the beginning of a word.
> matches an empty string, provided it is at the end of a word.
In regular expressions, the operators '*', '+', and '?' have the highest precedence, followed by
concatenation, and finally by '|'. As in arithmetic, parentheses can change how operators are
grouped.

Perhaps the best way of introducing the use of regular expressions in search is to work on some
search tasks. Since upper and lower case letters in Unix are different characters, they should be
taken care of in search. One way is to preprocess the text so that upper case letters are converted
to lower case while at the same time retaining the information that it is in fact an upper case
letter. One method is to put a special diacritic in front of the letter. In Swahili corpus such a
diacritic is asterisk '*'. If we want to catch the words with upper case first letter together with
those with lower case first letter, e.g. '*neema', and 'neema', we may write the query string:
' \*?neema '
This reads: Take away the special meaning of "*" (i.e. \*) and make its occurrence optional (?). Then find the string 'neema'.

Because of simplicity, however, in the following discussion and examples this is not taken into consideration and all normal words are assumed to have been written in lower case.

In the following, search tasks are discussed and queries formulated for extracting the needed constructions. The queries have also been tested with a small corpus of 32,270 words. The number of hits has been indicated after each query. For easy comparison, the number of hits found with SALAMA is given on the right side of '/'.

Task 2b. Extract deverbal nouns that end in -aji.

Here we have the same task as above in Task 2. Now we solve the task by using regular expressions. Thus:

```bash
tuuri$ cat hadithi.snt | sanalista | egrep ' (m|w|u)[a-z]*aji ' 
```

The command reads: Take the file hadithi.snt, put it to the format one-word-per-line, extract words that start with m, w, or u, followed by zero or more characters, followed by 'aji' and a blank.

This is far more simple than the earlier version. It is also possible to take context to the hit because the program is run in one piece. Thus:

```bash
tuuri$ cat hadithi.snt | egrep ' (m|w|u)[a-z]*aji ' 
```

Task 3. Search past tense verb-forms in affirmative from the corpus (Swahili).

The past tense marker is -li-, and it is located in the second morpheme slot in the verb structure. It is always preceded by a subject prefix, which may be any of the personal prefixes or noun class concords. We may try now to formulate the search string.

```bash
' (ni|u|a|tu|m|wa|i|li|ya|ki|vi|zi|ku|pa)li'     1,736 / 1,653
```

424:    <ADI> *walijifunza sana kutumia ulimi na midomo .$
1083: ataona nguvu ya mkono na uwezo wa ulimi wake .$
688: usu ya halmashauri hiyo palijengwa ulingo .$
519: > *kufumba kope na kufumbua mahali palipokuwa na majivu palitokea
chemchem

129: <ADI> *katikati_ya sakafu palitandikwa mazulia ya thamani kubwa .

519: fumbua mahali palipokuwa na majivu palitokea chemchemi ya maji safi .$

690: <ADI> *juu_yake palizunguka watu aliodhania kuwa mawazi

780: <ADI> *ulimwengu uliajabia dola yao kwa ushujaa wa askar

65: <ADI> *ugunduzi huu uliamsha wasiwasi mkubwa uliokuwa uking

1034: <ADI> *sasa ugomvi kati_yao ulianza juu_ya yule msichana mzuri aliy

The alternative subject prefixes are put within parentheses and separated with a vertical bar to indicate alternation. Before the left parenthesis '(' there is an empty space to indicate that it is the beginning of a word. The closing quote immediately after the string allows the word to continue, but it does not force it to do so. Because we are looking for verbs, the string has to continue. The following search string would ensure that the word continues at least one letter further.

' (ni|u|a|tu|m|wa|i|li|ya|ki|vi|zi|ku|pa)li[a-z]'

1,730 / 1,653

424: <ADI> *walijifunza sana kutumia ulimi na midomo .$

1083: ataona nguvu ya mkono na uwezo wa ulimi wake .$

989: ajini iliyokusanya utajiri wote wa ulimwengu .$

327: ambia ndugu zake kuwa kwa ushahidi uliotolewa , alitosheka kuwa baba yao a

686: > *ndani kulikuwa na uwanja mkubwa uliozungukwa na viti pande zote .$

873: <ADI> *ushirika katika ndoa ulipasa wanyama na ndege .$

Now there are words in Swahili that fulfil the given search pattern but are not verbs, e.g. niliye, walia, kilicho, etc., which are relative constructions. If we want to make sure that scanning proceeds beyond the end of the possible relative construction (e.g. nilicho), we have to modify the search string further.

' (ni|u|a|tu|m|wa|i|li|ya|ki|vi|zi|ku|pa)li[a-z][a-z][a-z][a-z]'

1,597 / 1,653

760: iokwenda , *adili alipoona *moyowe ulimshinda .$

989: ajini iliyokusanya utajiri wote wa ulimwengu .$

685: za hivyo alipita kwa mlango mkubwa uliozungukwa na viti pande zote .$

873: <ADI> *ushirika katika ndoa ulipasa wanyama na ndege .$

614: <ADI> *adili alikuwa na uchunguzi uliomwongoza katika ugunduzi mkubwa .$

327: ambia ndugu zake kuwa kwa ushahidi uliotolewa , alitosheka kuwa baba yao a
Because the longest relative morpheme (cho, vyo) is three characters long, we have to force the program to scan at least four characters forward. Even so it is not yet sure that the query will find only verbs, but now it excludes such relative constructions that do not contain a verb. But it still has the problem that it excludes single-character and two-character roots as (o-a, po-a, li-a, to-a, etc.). So we in fact should abandon the above search string and leave only two characters after the past tense marker -li-. So we make sure that we get all verbs but at the same time a lot of other words which fulfil the criteria. Thus:

' (ni|u|a|tu|m|wa|i|li|ya|ki|vi|zi|ku|pa)li[a-z][a-z]'  
1,716 / 1,653

Task 4. Search all tense/aspect verb-forms in affirmative from the corpus (Swahili).

Here the search task is much similar with the previous one. We have only to add all the tense/aspect markers as alternative. Thus:

' (ni|u|a|tu|m|wa|i|li|ya|ki|vi|zi|ku|pa)(na|ta|li|me|ki|ka|nge)[a-z][a-z]'  
2,323 / 2,431

We note that even the above search string is far from perfect, because it does not take care of such forms as -a-, hu-, and especially the lexicalised markers formed with the primary marker and some other verbal element, as the verb kwisha or a verb particle. Such forms include -mekwisha-, -meisha-, -mesha-, -sha-, -kisha-, -kiisha-, -lisha-, -liisha-, -ngeli-, -ngali-, -ngesha-, -japo-, and -taka-. Further, the marker -a- causes substantial changes to the subject prefix, and for this reason it has to be treated separately. The marker hu- does not allow subject prefixes at all, and also it has to be handled separately. Also we see that the number of compulsory characters after the marker has to be reduced to two, because -ki- and -ka- do not require the infinite marker ku- in the monosyllabic verbs.

We see that the task is complicated but not impossible to perform. Let us see each of the following search strings separately. We first modify the above string.

' (ni|u|a|tu|m|wa|i|li|ya|ki|vi|zi|ku|pa)(ni|na|ta|li|me|ki|
In searching for the constructions with the marker -a-, it is possible to construct the query in the following way:

' (na|wa|a|twa|mwa|ya|la|cha|vya|za|kwa|pa) [a-z] [a-z] [a-z] [a-z]' 2,218 / 63

But we see that it is very unreliable.

The query with the hu- marker is simple:

' hu[a-z][a-z]' 117 / 28

Note that here again the requirement of characters after the marker is two, because the infinitive marker ku- in monosyllabic verbs drops off. Also this query is unreliable.

When we compare the above search strings in regard to their accuracy we find that, in general, the longer the defined string is the more likely it is that it finds just the correct strings and the amount of wrong hits is small. And correspondingly the short query strings, such as the last one for hu-, is likely to extract a lot of wrong hits.
**Task 5. Extract all verb constructions with a relative marker in verb.**

There are two slots where the marker can be inserted, that is after the tense marker and the end of the verb. Let us see both of these cases separately.

Relative marker as a prefix:

The tense markers that allow the relative prefix marker are -na-, -li-, -taka-, and -si-. Thus the query string could be as follows:

```
'(ni|u|a|tu|m|wa|i|li|ya|ki|vi|zi|ku|pa)(na|taka|li|si)
(ye|o|yo|lo|cho|vyo|zo|ko|po|mo)[a-z][a-z]' 553
```

897: aliyageukia manyani akauliza kama yaliyosemwa yalitokea au hayakutokea .

$1233: > *matendo makubwa na bora kuliko yaliyotendwa zamani yangetendwa sasa ka

630: ipotaka kutoka nje aliona makabati yaliyotiwa nguo za hariri na sufu .$

903: na mianzi ya pua yake ilijaa maji yaliyozuia pumzi .$

55: limetia jina lake katika makumbusho yasiyosahaulika katika wakati wake , sa

719:                    <ADI> *alifuata zilikoongoza dalili mpaka akafika

78: likana kwa_sababu habari za *janibu zilikosekana .$

This query is fairly reliable, because the relative marker is now included into the search string. The shortest possible verb following the string is two characters long; so the maximum number of characters is two.

Relative marker as a suffix:

This case is much more complicated and practically impossible to perform with this method. The relative suffix allows only the subject prefix and optionally an object prefix in the construction. We may try to formulate the following query string:

```
'(ni|u|a|tu|m|wa|i|li|ya|ki|vi|zi|ku|pa)(ni|ku|m|mw|tu|wa|
u|i|li|ya|ki|vi|zi|pa|mu)?[a-z][az]+(ye|o|yo|lo|
cho|vyo|zo|ko|po|
mo)'   1,277
```

The above string reads:

```
'(ni|u|a|tu|m|wa|i|li|ya|ki|vi|zi|ku|pa)(ni|ku|m|mw|tu|wa|
u|i|li|ya|ki|vi|zi|pa|mu)?[a-z][az]+(ye|o|yo|lo|
cho|vyo|zo|ko|po|
mo)'   1,277
```

The above string reads:
Find in the beginning of a word a subject prefix, followed optionally by an object prefix, followed by two characters or more, followed by a relative suffix at the end of the word. Note that in front of the final quote there is an empty space to mark the end of the word.

This query string finds all the required word-forms. But how accurate is it?

We see that a total of 1,830 hits were found. SALAMA found a total of 487 cases only, as we see below.

Task 6. Extract all verb constructions with an object marker in verb.

This is already a non-trivial task, because the findings may help in finding out in which cases the object is marked in the verb although the object noun is also in the construction. The rule is that when the object is a human being the object will be marked also in the verb.

Because the object prefix is immediately before the verb root, the query string has to be formulated so that it includes all those morphemes that may occur before it. Here again, as in all cases above, the verb root itself cannot be identified in any way.

For affirmative constructions we have to write (at least) two query strings:

' (ni|u|a|tu|m|wa|i|li|ya|ki|vi|zi|ku|pa) (na|taka|li|si) (ye|o|yo|lo|cho|vyo|zo|ko|po|mo) (ni|ku|m|mw|tu|wa|u|i|li|ya|ki|vi|zi|pa|mu) [a-z][a-z]' 197

860: yokuwa nayo juu_ya bibi yule mzuri aliyempata katika *mji wa *mawe .$
752: itikia tu , lakini alitaja jina la aliyemwamkia vile_vile akamkaribisha kw
929: I> *malkia alimwuliza msichana mtu aliyemwokoa baharini alikuwa nani , na
1109: ikiri kwamba yeye peke_yake ndiye aliyepatwa na ajali mbaya kama ile
tang
1113: lakini hapakuwa na mtu hata mmoja aliyeuliza habari za ndugu zake .
793: onde moja , alione kana kama kiumbe kilichoumbwa maalum kuokoa kitu motoni
721: <ADI> kulikokuwa na chumba kidogo .
678: pozunguka nyuma_ya maduka alitokea kulikokuwa na nyumba moja kubwa kabisa
595: ona kwamba alikuwa si mtu ila jiwe lililokuwa halina maisha .

This query string extracts those constructions that have a relative prefix.

The remainder of the cases will be extracted with the following query string:

' (ni|u|a|tu|m|wa|i|li|ya|ki|vi|zi|ku|pa) (ni|na|ta|li|me|ki|ka|nge|mekwisha|meisha|mesha|sha|kisha|kiisha|lisha|liisha|ngeli|ngali|ngesha|japo) (ni|ku|m|mw|tu|wa|u|i|li|ya|ki|vi|zi|pa|mu)[a-z][a-z]' 788

1292: <ADI> *mfalme wa *majini alikubali ndugu zake kuwa watu tena .
288: wa kuzaliwa pacha hiyo mtoto mmoja alikufa pakabaki mmoja .
975: aliyotoswa baharini na ndugu zake alikuja upesi kumwokoa katika hatari .
939: <ADI> *adili alikubali kuwa alikumbuka kuona tandu jeupe , nyoka mw
910: , na kabla pumzi yake haijakwisha alikusudia kuwa watu tena .
805: <ADI> *kama umerogwa , *kabwere atawakomesha waliokuroga .
1190: ADI> *yakiokoka na kuwa watu tena atawapatanisha na ndugu yao waishi mais
867: <ADI> *walakini wakirudi *janibu atawatafutia mabibi wawili wazuri .

We see from the above that all this is quite complicated, and yet such forms as negative and subjunctive forms have been left out. Separate query strings are needed for those.

For part of negatives:

' (si|hu|ha|hatu|ham|hawa|hau|hai|hali|haya|haki|havi|hazi|haku|hapa)(ku|ta|nge|ngesha|ngeisha|ngeli|ngali) (ni|ku|m|mw|tu|wa|u|i|li|ya|ki|vi|zi|pa|mu)[a-z][a-z]' 27

959: <ADI> *na tangu alipotishia hivyo hakumpa nafasi hata ya uzi kupenyana tund
187: akati ule mpaka asubuhi , *ikibali hakupata usingizi kwa mawazo mbalimbali
194: ulikuwa mzito kwa mawazo , lakini hakuuliza neno juu_ya siri aliyogundua
In all, we had to write at least five query strings for extracting verb constructions with an object marker. The accuracy of such search strings is fairly good, but still far from perfect, because the verb root cannot be identified in any way.
We found a total of 1,730 hits (197+788+27+15+123). SALAMA found only 453 cases.

We now proceed to a still more complicated search task.

**Task 7. Extract such verb constructions that have an auxiliary verb kuwa as part of the structure (aliyekuwa akifanya, aliyekuwa anafanya, aliyekuwa amefanya).**

How would you perform this search task?
How reliable is the result?

(Hint: This can be done in two ways. Either, extract sentences with the auxiliary verb kuwa, and out of the result extract sentences with the main verb. Or, formulate a single query string that has both verbs)

The problem solved in two phases:

**Phase 1.**

The verb kuwa with relative prefix:

' (ni|u|a|tu|m|wa|i|li|ya|ki|vi|zi|ku|pa) (na|taka|li|si) (ye|o|yo|lo|cho|vyo|zo|ko|po|mo)kuwa ' 88

27: iri , na huruma yake juu_ya wanyama wasiokuwa wa mifugo ilishawishika sana
931: <ADI> *zamani maisha yake yalipokuwa katika hatari ya nyoka alipa
181: > *kisha yalifungwa minorororo kama yalivyokuwa .$
871: mba kulikuwa na mambo yaliyofaa na yaliyokuwa hayafai kushiriki .$
17: <ADI> *mazizi yaliyokuwa hayana wanyama walinjiliwa p

The verb kuwa without relative prefix (without -ki- and -ka- forms):

' (ni|u|a|tu|m|wa|i|li|ya|ki|vi|zi|ku|pa) (na|ta|li|me|nge|mekwisha|meisha|mesha|sha|kisha|kiisha|lisha|liisha|ngeli|ngali|ngesha|japo|singe|singali|singeli)kuwa ' 374

1320: <ADI> *mabibi wenyewe walikuwa wazuri na watanashati sana .$
1211: <ADI> *kama kesho nikiwaona wamekuwa watu tena nitakushukuru sana .

The verb kuwa without relative prefix, -ki- and -ka- forms:

' (ni|u|a|tu|m|wa|i|li|ya|ki|vi|zi|ku|pa)(ki|ka)wa '  14

The verb kuwa in subjunctive:

' (ni|u|a|tu|m|wa|i|li|ya|ki|vi|zi|ku|pa)(si)?we '  0

The verb kuwa in negative with negative prefix ha-:

' (si|hu|ha|hatu|ham|hawa|hai|hali|haya|haki|havi|hazi|haku|hapa)ku|ta|nge|ngesha|ngeisha|ngeli|ngali)kuwa '  0

Note that all these operations have to be done from the original file, and they should not be run in a pipe. The results of each run should be concatenated as one file.

Phase 2.

In the next phase, query strings for the main verb are formed and the results of the first phase are given to it as input.

' (ni|u|a|tu|m|wa|i|li|ya|ki|vi|zi|ku|pa)(na|ki|me)(ni|ku|m|mw|tu|wa|u|i|li|ya|ki|vi|zi|pa|mu)?[a-z][a-z]'  241

It is assumed that if a hit fulfilling these search criteria is found, it has a good chance to be part of the required construction. What is the major problem with this assumption?

The problem solved in one phase:

Now we try to formulate a series of queries that will find the required constructions in one phase. This is done by adding the query string for the main verb to the query string of the verb kuwa.
Thus:

'+ (ni|u|a|tu|m|wa|i|li|ya|ki|vi|zi|ku|pa)(na|ta|li|me|nge|
mekwisha|meisha|mesha|sha|kisha|kiisha|lisha|liisha|ngeli|
gali|nesha|japo|singe|singali|singeli)kuwa (ni|u|a|tu|m|wa|
 i|li|ya|ki|vi|zi|ku|pa)(na|ki|me)(ni|ku|m|mw|tu|wa|u|i|li|ya|ki|
vi|zi|pa|mu)?[a-z][a-z]' 13

1131: e ya moto yalitoka kinywani mwake alipokuwa kiamru hili , na mara neno
1 859:         <ADI> *wakati alipokuwa akisema na ndugu zake
stahani
64:         <ADI> *siku moja *rai alipokuwa akitazama hesabu za manyama
w
882:         <ADI> *usiku *adili alipokuwa amelala na jahazi inakwenda
m
1221: I> *alitingisha chwa chake kama aliyekuwa akikataa wazo la kukata
tamaa
1128:         <ADI> *alimrudia *adili aliyekuwa anaanza kupata fahamu
akasema
65: unduzi huu uliamsha wasiwasi mkubwa ulikuwa ukingojea mguso mdogo tu
katik
66:         <ADI> *moyo wake ulipokuwa ukiliwa na wasiwasi ,
mawazo
658: *vitu vilivyobaki vilimtazama kama vilivyokuwa vikisimanga choyo chake .$
307:         <ADI> *watu walinyamaza kama waliokuwa wamenyang'anywa ndimi zao .$
323:         <ADI> *wageni walipokuwa wanakwenda zao waliambiana
n
271: wanza yaliinua juu mikono yao kama yaliyokuwa yakiomba .$

' (ni|u|a|tu|m|wa|i|li|ya|ki|vi|zi|ku|pa)(na|ta|li|me|nge|
mekwisha|meisha|mesha|sha|kisha|kiisha|lisha|liisha|ngeli|
gali|ngesha|japo|singe|singali|singeli)kuwa (ni|u|a|tu|m|wa|
 i|li|ya|ki|vi|zi|ku|pa)(na|ki|me)(ni|ku|m|mw|tu|wa|u|i|li|
ya|ki|vi|zi|pa|mu)?[a-z][a-z]' 32

1235: ewe aliwasamehe ; na *mfalme *rai alikuwa akijaribu kuwaokoa katika
balaa
717:         <ADI> *kwanzu alikuwa akipanda kwa kusitasita ,
lakin
142:         <ADI> *ikibali alikuwa akisanifu mashairi , kama
ilivy
99: DI> *adhuhuri ilipokaribia *ikibali alikuwa amefika katika viunga vya mji
a
351:         <ADI> *siku moja *adili alikuwa amekaa dukani pake .$
1226: I> *aliendelea kufikiri kuwa *rai alikuwa ametukuzwa kwa ufalme na
hekima
There queries should do the same as the queries carried out in two phases. In what ways are the results different? Which approach brings better results?

### 1.3 Tools based on linguistic analysis

We found above that regular expressions are quite useful and efficient in formulating such query strings which otherwise would require a large number of separate queries. We also noted that the accuracy of the queries increases when there are several features available for defining the verb, and with few features the error rate increases sharply.

The recall and precision of the queries, even of complicated ones, will reach an entirely different level if we have access to a system that performs first the linguistic analysis of the text. As we said above, the same result can be achieved if we have a tagged corpus as source material. Below we shall see how the tasks described above can be performed with the system that includes a linguistic analysis.

**Task 3b. Search past tense verb-forms in affirmative from the corpus (Swahili).**

The search string is simply:
Task 4b. Search all tense/aspect verb-forms in affirmative from the corpus (Swahili).

The search string is:

' VFIN '

The result contains also negative forms. To get rid of them one needs to filter them out. The tag for negative forms is 'NEG'.

The tag VFIN means any verb that is in finite form. If one wants to limit the search to selected tense/aspect forms, one just writes those tags into the query. For example, the -na-, -a-, -me-, -li- and -ta- forms may be retrieved with the following query strings:

' PR:na '
' PR:a '
' PERF:me '
' PAST:li '
' FUT:ta ' 

file:///D|/Documents/salama/corptrain.htm (20 of 35)24.11.2004 16:53:50
And the combined query is:

' (PR:na|PR:a|PERF:me|PAST:li|FUT:ta) ' 

polva$ cat hadithi.snt | run9-snt | egrep ' (PR:na|PR:a|PERF:me|PAST:li|FUT:ta) ' 

"<kimeandikwa>" "andika" 7/8-SG-SP VFIN PERF:me V SV SVO SVOO ' write ' PASS & 
"<alikuwa>" "wa" 1/2-SG3-SP VFIN PAST INFMARK ' be ' V SV INTR MONOSLB & 
"<*alikuwa>" "wa" 1/2-SG3-SP VFIN PAST INFMARK ' be ' V SV INTR MONOSLB & 
"<iliyohitilafiana>" "hitilafiana" 9/10-SG-SP VFIN PAST 9/10-SG REL V AR SV ' differ, be different' & 
"<ili jigawa>" "gawa" 9/10-SG-SP VFIN PAST REL-SP OBJ V SV SVO ' divide, distribute' & 
"<yanapendwa>" "penda" 5/6-PL-SP VFIN PR:na V SV SVO AUXMOD ' love ' PASS & 
"<alikolewa>" "okolea" 1/2-SG3-SP VFIN FUT:ta V SV SVO APPL PASS & 

Task 5b. Extract all verb constructions with a relative marker in verb.

The search strings are:

' REL ' 
' NEG-REL ' 

And the combined query is:

' (REL|NEG-REL) ' 

Because the tag 'REL' is also a tag for such relative constructions that are not verbs, we have to make sure that the word is a verb. The following command does the job:

polva$ cat hadithi.snt | run9-snt | egrep ' V ' | egrep ' (REL|NEG-REL) ' 

"<iliyoghadhibika>" "ghadhibika" 3/4-PL-SP VFIN PAST 5/6-PL REL V AR SV STAT & 
"<alipolala>" "lala" 1/2-SG3-SP VFIN PAST 16-SG REL V SV HC & 
"<aliyeokolewa>" "okolea" 1/2-SG3-SP VFIN PAST 1/2-SG REL V SV SVO APPL PASS & 
"<yalikutana>" "kutana" 5/6-PL-SP VFIN PAST 16-SG REL V SV SVO HC REC & 
"<iliyopoteana>" "poteana" 3/4-PL-SP VFIN PAST 9/10-SG REL V SV REC & 
"<alipomwambia>" "ambia" 1/2-SG3-SP VFIN PAST 16-SG REL 1/2-SG3 OBJ V SV SVOO APPL &
Task 6b. Extract all verb constructions with an object marker in verb.

The search string is:

' OBJ '  

polva$ cat hadithi.snt | run9-snt | egrep ' OBJ '  

"<alimshukuru>" "shukuru"  1/2-SG3-SP VFIN PAST 1/2-SG3 OBJ V AR SV SVO &  
"<walimwona>" "ona"  1/2-PL3-SP VFIN PAST 1/2-SG3 OBJ V SV SVO ' see, feel ' &  
"<*waliwatambua>" "tambua"  1/2-PL3-SP VFIN PAST 1/2-PL3 OBJ V SV SVO  
"<wakamwombea>" "ombea"  1/2-PL3-SP VFIN NARR:ka 1/2-SG3 OBJ V SV SVO ' ask for, beg' APPL &  
"<aliwavika>" "vika"  1/2-SG3-SP VFIN PAST 1/2-PL3 OBJ V SV  

Task 7b. Extract such verb constructions that have an auxiliary verb kuwa as part of the structure (aliyekuwa akifanya, aliyekuwa anafanya, aliyekuwa amefanya).

Surprisingly, this task is not simple to perform in this environment, because we have to handle two separate words at the same time. The problem is that we handle here a file that is in the format of one-word-per-line. If we extract the verb-forms of kuwa and then the main verbs, we cannot directly make sure that they belong to the same construction. However, we may solve the problem in the following way:

1. Extract the verb-forms of kuwa.

Search strings:

' V '  
' "wa" '  

Run these in a pipe. Extract first the verbs and out of them the verb-forms of kuwa. Define the context so that the line after the hit will be included.

2. Extract the main verbs from the result of the previous run.

Search strings:

' COND:ki '  
' PR:na '  
' PERF:me '  


The combined query is:

' (COND:ki|PR:na|PERF:me) '

Define the context so that the line before the hit will be included. The result should contain lines with the verb kuwa, each followed by a line with the main verb.

The command line for the whole operation would be:

```
polva$ cat hadithi.snt | run9-snt |
egrep ' V ' |
cgrep -B 0 -A 1 ' ("wa") ' |
cgrep -B 1 -A 0 ' (COND:ki|PR:na|PERF:me) ' 63
>>> "<ilikuwa>" "wa" 9/10-SG-SP VFIN PAST INFMARK ' be ' V SV INTR MONOSLB &
>>> "<imetumika>" "tumika" 9/10-SG-SP VFIN PERF:me V SV SVO STAT &
--------
>>> "<alikuwa>" "wa" 1/2-SG3-SP VFIN PAST INFMARK ' be ' V SV INTR MONOSLB &
>>> "<amekaa>" "kaa" 1/2-SG3-SP VFIN PERF:me V SV ' sit, be in place ' &
--------
>>> "<*walipokuwa>" "wa" 1/2-PL3-SP VFIN PAST 16-SG REL INFMARK ' be ' V SV INTR MONOSLB &
>>> "<inakwenda>" "kwenda" 9/10-SG-SP VFIN PR:na INFMARK V SV &
--------
>>> "<*alipokuwa>" "wa" 1/2-SG3-SP VFIN PAST 16-SG REL INFMARK ' be ' V SV INTR MONOSLB &
>>> "<akitoa>" "toa" 1/2-SG3-SP VFIN COND:ki V SV SVO &
--------
>>> "<aliyekuwa>" "wa" 1/2-SG3-SP VFIN PAST 1/2-SG REL INFMARK ' be ' V SV INTR MONOSLB &
>>> "<anaanza>" "anza" 1/2-SG3-SP VFIN PR:na V SV SVO ' begin ' &
```

Note that here we use two versions of grep, i.e. egrep (extended grep), and cgrep (grep which allows the definition of context as number of lines before and after the hit). -B stands for 'before the hit' and -A stands for 'after the hit'.

So far we have taken examples of verb constructions and demonstrated that with the help of regular expressions it is possible to construct query strings that find those constructions from text. However, the level of recall and precision varies greatly depending on the type of task. We have also seen that language analysis enhances the search tasks tremendously. And what is more important, it increases accuracy considerably.

Now we shall take such examples where the word structure is not the key problem.
Task 8. Extract (mostly idiomatic) constructions with the verb *piga* (*piga* picha, *piga* pasi, etc.).

We have to search for such collocations where the first element is the verb *piga* and the second element is a noun. The assumption is that this is the most basic construction. There might be a possibility that there are more than two elements in the construction, although those cases are not common. Because the verb *piga* 'to hit' is quite common in its primary use, we should exclude those cases. One could assume that by leaving out those cases where the object prefix is part of the verb-form primary uses would be excluded. The close examination shows, however, that although it excludes them, it does not exclude them all. And what is more fatal, it also excludes the cases where the object prefix is part of the structure although it is in idiomatic use. So it is better to accept all the verb-forms of *piga*.

Query strings:

' V '  
' ("piga") '  

These have to be run in pipe.

The command would thus be:

```
polva$ cat sourcefile | anal-word-per-line |  
cgrep -B 0 -A 3 ' V ' |  
cgrep -B 0 -A 3 ' ("piga") ' |  
cgrep -B 1 -A 2 ' (N|ADV) '  
```

In the above command, the sourcefile is first analysed. Then verbs with three following words are extracted. Out of this the verb-forms of the verb *piga* are extracted, together with three following words. Finally nouns and adverbs with one preceding and two following words are extracted. The last query is based on the assumption that normally the verb is followed by the noun that is part of the structure, or occasionally by an adverb, which may come between the verb and the noun. However, this query is not fully reliable, because it might restrict the occurrence of more complicated structures.

Perhaps the following command is more reliable although it allows for more unwanted hits.

```
polva$ cat sourcefile | anal-word-per-line |  
cgrep -B 0 -A 4 ' V ' |  
cgrep -B 0 -A 4 ' ("piga") '  
```

44
Task 9. Extract all words of Arabic and Indian origin.

In performing this task it is necessary to have either (a) an analysis program in use or (b) have a corpus where these features have been tagged. Such etymological information is available seldom even in tagged corpora. In SALAMA this operation is very easy, because each word root is marked for etymological information.

The following command string does the job and extracts the words of Arabic and Indian origin in the order where they occur in the source file.

```bash
polva$ cat hadithi.snt | run9-snt | egrep ' (AR|IND) ' 2,761
```

```plaintext
"<namna>" "namna" 9/10-0-SG N IND B ' kind, way ' &
"<duniani>" "dunia" 9/10-0-SG N AR ' world ' LOC &
"<rangi>" "rang" 9/10-0-SG N IND B ' colour ' &
"<tabia>" "tabia" 9/10-0-SG N ' habit ' AR &
"<iliyohitilafiana>" "hitilafiana" 9/10-SG-SP VFIN PAST 9/10-SG REL V AR SV ' differ, be different ' &
"<kabisa>" "kabisa" AD-ADJ ' totally, wholly ' AR @<A-AD &
"<tabia>" "tabia" 9/10-0-PL N ' habit ' AR &
"<zamani>" "zama" 9/10-0-PL N AR ' ancient, old times ' LOC &
"<kodi>" "kodi" 9/10-0-SG N IND B &
```

If we want both types into separate files, we may use the following commands:

```bash
polva$ cat sourcefile | anal-word-per-line |
egrep ' AR ' > ar.res 2,681
```

```bash
polva$ cat sourcefile | anal-word-per-line |
```
2. Work with tagged corpora

So far we have been working with files with plain text. No tagging has been assumed to be in them. Normally, although the text does not have POS tags or more sophisticated tagging, it has some structural information that can be utilised in information extraction. For example, it is often important to know whose text is in question, when it was produced, from which page in the book or in a newspaper it is, who is the speaker in a transcribed text, from whose line is the piece of text in a drama, etc. This information can be coded in many ways. There are programs that pick the information from any place in the file. Background information is normally located at the beginning of the file, and the background info needed to be linked to the search result can be retrieved from there. Another method is to place identification codes in suitable places, for example at the beginning of each line. This method ensures that practically all retrieval tools are able to print the code.

Examples of encoding systems in Helsinki Corpus of Swahili:

Extract from the file books-all:

```
<UJA> *ujamaa *julius *k. *nyerere *oxford *university *press
*dar_es_*salaam , 1968 ( *reprinted 1974 )$
<UJA> *utangulizi $
<UJA> *tangu mwaka 1962 , *t*a*n*u imeamua rasmi kujenga taifa la *ujamaa .$
<UJA> *lakini kwa mrefu maana ya *ujamaa katika mazingira ya *tanzania
haikuelezwawazi wazi .$
<UJA> *kijitabu kile cha ' *ujamaa ' , kilichotolewa mwaka 1962 , kimeeleza
mawazo ya *ujamaa kwa jumla , lakini kiliandikwa kwa *kiingereza , kwa_hyo
kwa_kweli hakikuwafikia wananchi wa *tanzania kwa urahisi .$
<UJA> *kwa_hyo viongozi wengi wa *t*a*n*u , na vile_vile walimu na wafanya
kazi wengi wa *serikali , hawakuelewa vema hata ile_misingi ya maana ya
*ujamaa ambayo walikuwa na wajibu wa kuitetea na kuitekeleza .$
```

Each book in the corpus has a unique code, and this code is repeated in the beginning of each sentence. The books are preprocessed so that each sentence is on its own line. Also other kinds of modifications have been done to the text, to make it more suitable for automatic processing. The page numbers of original books have also been retained, and they can be retrieved to the search result.

Extract from the file art-all:
CDA CAMEEL dissemination week

[lengo 88 *desemba namba 189]
*maoni *ya *mhari $
*baada_ya kungojea kwa karne nyingi , siku ilifika ambayo ilitimia *ahadi ya
*mwenyezi *mungu,ya kumleta duniani *mwanae *mpendwa , *yesu *kristo ,
kutukomboa sisi binadamu kutoka makucha ya shetani yaliyotunasa tangu wazazi
wetu wa kwa kwanza , *adamu na *eva , walipotumbukia katika dimbwi la dhambi
na kufukuzwa katika bustani ya *paradiso .$
*siku hiyo ilifika karibu miaka elfu mbili ( 2,000 ) iliypita alipozaliwa
*mkombozi wetu *yesu *kristo katika nchi ya *wayahudi .$
*viongozi wetu wa kwanza wa *kanisa waliita siku hiyo " *krismasi "
inayokumbukwa na kusherekewa kote ulimwenguni na *wakristo na wengine wario
wakristo ifikapo tarehe 25 *desemba ya kila mwaka .$
*kwetu sisi *wakristo , bila kujali tofauti yetu ya madhehebu , *sikukuu hii
ya *krismasi ni kubwa sana kiroho kwa_kwa inatukumbusha *upendo na *huruma
ya *mwenyezi *mungu ambaye kwa upendo wake alituhurumia sisi binadamu
 tuliomwasi na kuwachungu wa shetani kwa kumleta duniani *mwena wake
kutukomboa kutoka utumwa wa shetani na kutufungulia tena mlango wa *paradiso .$
$*aidha , siku hiyo ya *krismasi ni kubwa sana *kwetu wakristo , kama
tunavyoamini , *yesu *kristo anarudi na kuzaliwa rohoni mwetu .$

This file contains newspaper articles from several newspapers. The background info is in the
beginning of each article, and it can be retrieved to the search result.

Extract from transcriptions of tapes:

Edited by TR, December 1996
DAHE 25
PLACE: Makunduchi
Musa (HC), 7. Mohammed Abdallah Dume (MD), 8. Muhammed bin Fattawi bin Issa
(MF), 9. Muhammed Abdul Haji (MH), 10. Hassan Abassi (HA), Anonyme (XX)
INTERVIEWER(S): T.S.Y. Sengo (TS)
RECORER: Sony Professional
SPEED:
LANGUAGE: Kikae (Kimakunduchi)
TOPIC: 1. Story about a king and horses, 2. Story about a man named Makame,
3. Story about a king, 4. Another story about a man named Makame, 5. Another
story about a king, 6. Story about a man named Makame, 7. Story about a poor
man, 8. Native medicine in every day life
SIDE A
Ki TS M Anza.
Ka AY F Jina lyangu Amina, Amina Yusufu, Yusufu Asani.
Ki TS M Sema tu.  
Ka AY F Nganani, Kae. Paukwa.  
Ki TS M Pakawa.  
Ka HC M Bustani ya yule mfalme.

These files contain discussions by groups of people (2 or more). The background info is located in the beginning of each discussion event and it can be attached to the search result. Each speech event is marked for the speaker, the sex, and the language/dialect. It is, therefore, possible to handle the material according to the speaker, the sex, and the variety of speech that the speaker represents.

Extracts from word-lists:

PLACE: Kibiti Secondary School  
Language: [Kizigua]  
TIME: 8/9/1991  
NOTE: numbers 031, 036, 072, 220, 267, 275 possibly word initial syllabic nasal  
001 ruf *mwili | do  
002 ruf *tumbo | nda, zinda  
003 ruf *kichwa | mtwi, mitwi  
004 ruf *ubongo | do  
005 ruf *nywele | fili  
006 ruf *kipaji | ubala (?)  
007 ruf *uso | do  
008 ruf *shavu | funda, izifunda  
009 ruf *taya | ulusaya, izisaya

PLACE: Utete  
TIME: 3.9.1991  
NAME: Mohamedi A. Kilungi, (dialect: Kimatumbi)  
PLACE: Mafia  
TIME: 16.9.1991  
Language: [Kimatumbi]  
001 mat *mwili | do  
002 mat *tumbo | ndimbo  
003 mat *kichwa | mtuwi, mituwi
Word-lists contain glosses of 640 items in various dialects and languages. These lists are meant primarily for comparative research. Each item has a unique number. In addition to this, each item has a code for the informant and a language/dialect code. Therefore, it is possible to extract the required piece of information by using various codes successively as filters.

Extracts from parliament discussions:


Kwa nini sasa kama Serikali inaweza kusimamia hata kipande cha gazeti la Majira, kwako mpaka leo inashindwa Serikali kukusanya kodi, kuwabana hawa ambao kuwa hawalipi kodi, hebu jamani tujiulize. Nguvu hii itumike upande mmoja tu. Nani analipa kodi, ni wananchi wetu na wananchi si wanaolipa kodi. Sasa hawa tax payers wanatarajia kupata nini. Wanatarajia ku-benefit from what they are contributing.


These are transcriptions of discussions and speeches in the Tanzanian Parliament from 1996. Encoding is still very minimal, and more work should be done with it. The transcription has an
interesting feature in that the English words used in speech have been specially marked with '-'.

Helsinki Corpus of Swahili is not a tagged corpus in the sense that the term normally means. It is in no way balanced, and it contains very different kinds of materials. Actually it is an archive of Swahili language, and new material is included into it as far as resources allow.

The material in the corpus is in the format that it allows the work on all three levels discussed above, i.e. the direct string search, the string search enhanced with regular expressions, and the use of language analyser, or tagger, if you like.

**Exercises**

1. Find all negative verb-forms in the Swahili corpus.

2. Find all pronouns of the Swahili corpus.

3. Find the verbs with an objective marker and the verbs with the reciprocal marker.

4. Find the nouns of the type ‘mwanashamba’, with -ana- as a distinctive feature, from the corpus.

5. What percentage of words in Kezilahabi’s books is of Arabic origin?

   (Hint: Kezilahabi’s books have a code which starts <KEZ-…>. The code is in the beginning of every line.)

6. On which pages does the word mtumishi (pl. watumishi) occur in Kezilahabi’s Rosa Mistika?

   (Hint: The book has a code <KEZ-ROS>. The code is in the beginning of every line.)

7. In which newspaper articles does the term UKIMWI appear?

   (Hint: Each article has a code in the beginning, such as: [Majira, May 15, 2001])

8. Retrieve all the cases when Tigiti Sengo uses the word 'ujaji'?

   (Hint: Transcriptions have a unique code for Sengo: <TS>. It is in the beginning of each speech section of Sengo.)

9. Retrieve all the variants of the Swahili words 'jicho', 'sikio', 'tumbo' and 'nywele' from the word-
lists.

Formulate the task first in words and then formulate the actual query string. Evaluate the performance of the query string(s) in terms of (a) recall and (b) precision.

**Solutions to exercises:**

1. Find all negative verb-forms in the Swahili corpus.

```
' (si|hu|ha|hatu|ham|hawa|hau|hai|hali|haya|haki|havi|hazi|haku|hapa)
  (ku|ta|nge|ngesha|ngeisha|ngeli|ngali) (ni|ku|m|mw|tu|wa|u|i|li|ya|ki|
  vi|zi|pa|mu)?[a-z][a-z]'
```

Here the object marker is redundant, because it is made optional and it therefore does not constrain the search in any way.

```
' (ni|u|a|tu|m|wa|i|li|ya|ki|vi|zi|ku|pa) (si|singe|singeisha|singesha|
  singeli|singali) (ni|ku|m|mw|tu|wa|u|i|li|ya|ki|vi|zi|pa|mu)?[a-z]
  [a-z]+(e|i|u) '
```

Here also the object marker is redundant.

```
' (ni|u|a|tu|m|wa|i|li|ya|ki|vi|zi|ku|pa) si(ye|o|yo|lo|cho|vyo|zo|ko|
  po|mo)? (ni|ku|m|mw|tu|wa|u|i|li|ya|ki|vi|zi|pa|mu)?[a-z][a-z]+(e|i|u) '
```

Here both the relative marker and the object marker are made optional. Both are redundant. In fact the whole search string is redundant, because all the occurrences will be found with the previous search string.

```
' (si|hu|ha|hatu|ham|hawa|hau|hai|hali|haya|haki|havi|hazi|haku|hapa)
  (ni|ku|m|mw|tu|wa|u|i|li|ya|ki|vi|zi|pa|mu)?[a-z][a-z]+(i|e|u) '
```

Here object marker is optional and therefore redundant. The remaining string has so few features that it is likely to produce plenty of rubbish.

```
' kuto(ku)?[a-z][a-z]+ '
```

In sum, there are no sufficiently efficient methods for doing the task with string search enhanced with regular expressions.
2. Find all pronouns of the Swahili corpus.

' (mimi|wewe|ye|sisi|ny?inyi|wao) ' 
' (w|y|l|ch|vy|z|kw|p|mw) (angu|ako|ake|etu|enu|ao) ' 
' h(uyu|awa|uu|ii|ili|aya|iki|ivi|izi|uku|apa|umu) ' 
' h(uyo|ao|uo|iyo|ilo|ayo|icho|ivyo|izo|uko|apo|umo) ' 
' (yu|wa|u|i|li|ya|ki|vi|zi|ku|pa|mu?)le ' 

3. Find the verbs with an object marker and the verbs with the reciprocal marker.

First verbs with object marker:

Relative marker + object marker:

' (ni|u|a|tu|m|wa|i|li|ya|ki|vi|zi|ku|pa) (na|taka|li|si) (ye|o|yo|lo|cho|vyo|zo|ko|po|mo) (ni|ku|m|mw|tu|wa|u|i|li|ya|ki|vi|zi|pa|mu?|ji) [a-z][a-z]+ ' 

Object marker:

' (ni|u|a|tu|m|wa|i|li|ya|ki|vi|zi|ku|pa) (na|tal|me|ki|ka|nge|mekwisha|meisha|mesha|sha|kisha|kiisha|lisha|liisha|ngeli|ngali|ngesha|ja|pa) (ni|ku|m|mw|tu|wa|u|i|li|ya|ki|vi|zi|pa|mu|ji) [a-z][a-z]+ ' 

Negative SP + object marker:

' (si|hu|ha|hatu|ham|hawa|hai|hali|haya|haki|havi|hazi|haku|hapa) (ku|ta|nge|ngesha|n|ja|sha|ngeli|ngali) (ni|ku|m|mw|tu|wa|u|i|li|ya|ki|vi|zi|pa|mu|ji) [a-z][a-z]+ ' 

Negative T/A-marker + object marker:

' (ni|u|a|tu|m|wa|i|li|ya|ki|vi|zi|ku|pa) (si|singe|singeisha|singesha|singeli|singali) (ni|ku|m|mw|tu|wa|u|i|li|ya|ki|vi|zi|pa|mu) [a-z][a-z]+ ' 

Negative SP + object marker, without T/A-marker:

' (si|hu|ha|hatu|ham|hawa|hai|hali|haya|haki|havi|hazi|haku|hapa) (ni|ku|m|mw|tu|wa|u|i|li|ya|ki|vi|zi|pa|mu) [a-z][a-z]+(i|e|u) '

Then find verbs with reciprocal marker:

With relative marker:

' (ni|u|a|tu|m|wa|i|li|ya|ki|vi|zi|ku|pa) (na|taka|li|si) (ye|o|yo|lo|cho|vyo|zo|ko|po|mo) [a-z]+(ana) [a-z]?[a-z]?[a-z]? '

Without relative marker:

' (ni|u|a|tu|m|wa|i|li|ya|ki|vi|zi|ku|pa) (na|ta|li|me|ki|ka|nge|mekwisha|meisha|mesha|sha|kisha|kiisha|lisha|liisha|ngeli|ngali|ngesha|japo) [a-z]+(ana) [a-z]?[a-z]?[a-z]? '

With negative SP + T/A-marker:

' (si|hu|ha|hatu|ham|hawa|hau|hai|hali|haya|haki|havi|hazi|haku|hapa) (ku|ta|nge|ngesha|ngeisha|ngeli|ngali) [a-z]+(ana) [a-z]?[a-z]?[a-z]? '

With negative T/A-marker:

' (ni|u|a|tu|m|wa|i|li|ya|ki|vi|zi|ku|pa) (si|singe|singeisha|singesha|singeli|singali) [a-z](ani)[a-z]?[a-z]?[a-z]? '

With negative SP, no T/A-marker:

' (si|hu|ha|hatu|ham|hawa|hau|hai|hali|haya|haki|havi|hazi|haku|hapa) [a-z]+(ani) ? '

4. Find the nouns of the type ‘mwanashamba’, with -ana- as a distinctive feature, from the corpus.

' (mw|w)ana[a-z][a-z]+ ' 

5. What percentage of words in Kezilahabi’s books is of Arabic origin?
First, take the file books-all. Then run the program wlist, which verticalises the text and deletes punctuation marks and numbers. Then count lines with the utility count-lines.

Second, take the file books-all. Extract the lines that include one of the codes of Kezilahabi’s books. Then analyse the text with run9-snt, which expects that the text is in sentence-per-line format, and which produces the output where a word and the result of analysis are on the same line. Extract the lines that contain the tag AR. Then count lines with the utility count-lines.

(Note: You cannot use wc here. Each hit contains several ‘words’, because also the codes are considered as words in the wc utility, and we want to count only the words with Arabic origin.)

Then calculate the percentage on the basis of these two numbers.

Note that you will get the percentage of the total occurrence of Arabic words. If you want to get the list of various word-forms only, you should do more filtering to both query strings. For example:

Then calculate the percentage on the basis of these two numbers.

Note that you will get the percentage of the total occurrence of Arabic words. If you want to get the list of various word-forms only, you should do more filtering to both query strings. For example:

6. On which pages does the word mtumishi (pl. watumishi) occur in Kezilahabi’s Rosa Mistika?

Take the file books-all. Extract the lines that contain the required string and put the page number (surrounded by square brackets, in the footer) in front of the line. Extract the lines that contain the tag <ROS>. Cut the page number as a separate line. Extract the lines containing '\['. Sort them and delete duplicates (in case the string occurs on the page more often than once).

7. In which newspaper articles does the term UKIMWI appear?

Take the file art-all. Extract the lines that contain the term UKIMWI, with the page number in the header. Cut the page number on its own line. Extract the lines that contain '\['. Sort the lines and
8. Retrieve all the cases when Tigiti Sengo uses the word 'ujaji'?

```
polva$ cat dahe-all | egrep '<TS>' | egrep 'ujaji'
```

Take the transcription file dahe-all | Extract the lines which contain the tag <TS>. Extract the lines which have the term ujaji, also those terms that are in the beginning of the line.

9. Retrieve all the variants of the Swahili words 'jicho', 'sikio', 'tumbo' and 'nywele' from the word-lists.

```
polva$ cat wlist-all | egrep '(013|015|002|005)'
```

Take the file wlist-all. Extract the line numbers that contain the required words.