Search engines and linguistics

– with a case study of an automated compound translator using search engines

Hanne Moa

Abstract

The fundamental problem of machine translation is that in order to translate, real world knowledge is needed. Real world knowledge can be emulated by a sufficiently large corpus, and the Internet can be viewed and used as an extremely large corpus.

An important problem when translating from Norwegian and many other languages is what to do with compounds. It has been shown before that emulating real world knowledge with a corpus works well for compounds; compound-translation having been a showcase of using the Internet as a corpus for translation (Grefenstette, 1999).

However, the Internet and how to access it changes, adapting to its users. Linguists using the Internet for research must likewise adapt. This thesis therefore has two goals: to show how to use current search engines to treat the Internet as a corpus for doing linguistics, and building on that, to show how the search engines of today can be used to help translate compounds.

This thesis presents `ReCompounder`, a working prototype for non-supervised, fully automatic translation of compounds by using the web as language model and a comprehensive bilingual dictionary as translation model. Evaluation shows that this strategy is viable with current search engine technologies and that the produced translations are good, with better coverage than the method from Grefenstette 1999. Furthermore, it has proven useful for lexicography by increasing the amount of and checking the information about existing lexical items.

Acknowledgements

Thanks go to my advisor Torbjørn Nordgård, if I didn't run down your doors it was because I was too busy putting thoughts in the right order before putting them down on paper. Then there's the library-gang of fellow merry grad-students (and LaTeX-users) and the rest on the fifth floor of building 4 (Heidi, how do you find the time to work on your dissertation when you're always organizing things for us?)

Funding for a year was provided by the LOGON project, but more important than that, LOGON provided the secondary linguistic resources that was necessary for this project, and its members provided discussion, pointers and intellectual support.

Thanks go to Daniel Eugene Lacey-Mcdermott for the evaluation, we never did get around to that evening of beer so do get in touch! Oh, and thanks, Jonathan, for finding Daniel: I do agree that studying linguistics does weird things to one's head.

Thanks go to the Nordic Graduate Schools of Linguistics-system and the class I attended on linguistic resources, a discussion there triggered the slap-hand-on-forehead eureka of how to access XML easily and generally.

Thanks to the organizers of NODALIDA 2005 for providing lovely food, nice weather and a time and place to meet and discuss computational linguistics. For some reason I especially remember the phonetics of Chimay and how to explain Norwegian /y/ to a non-Norwegian and the other interesting effects of phonetics on beer, proving once and for all that phonetics is bad for you.

Last but not least, I thank my husband and test-reader Steinar, and my grandparents for their joy of reading.

Contents

List of Figures

LIST OF FIGURES

List of Tables

LIST OF TABLES

Listings

Chapter 1

"Text must be (minimally) understood before translation can proceed effectively. Computer understanding of text is too difficult. Therefore, Machine Translation is infeasible."

(Bar-Hillel, 1960)

Introduction

It is still not clear what it means that a human being *understands* a text. Little wonder then that a computer still cannot understand a text in any meaningful definition of the word *understand*. When translating, be it by machine or by hand, it is sooner or later necessary to look up a word in the source language and see what it translates to in the target language, by using a bilingual dictionary or traversing a neural net or searching in a list of known good alignments. It is equally certain that no such collection of translation-pairs can contain all possible words of a language[1] or all possible translations of a word.

Unknown, previously unseen or unregistered words are not all of a kind, though. In some cases it is possible to cheat. One such case is words that are names. They can usually be passed through unchanged[2]. Semantically transparent compounds is another case.

A *transparent compound* contains its own head[3] and this head is modified by the other words or stems in the compound. It is then possible to attempt translation by translating each stem and reassembling them in a form suitable to the source language.

The original purpose of this thesis was to demonstrate a compound trans-

[1]Unless it is well and truly fossilized like Latin.

[2]Unless it is a name of a geographical feature a native user of the target language cannot be expected to know.

[3]Or heads, in case of dvandva-compounds like *singer-songwriter* or *bittersweet*.

lator that translates from Norwegian to English, but as is often the case, the questions along the road proved more interesting than the goal. For instance, how a compound is analyzed prior to translation, where to get the stem-translation pairs, how to know which translations are at all possible, and how to check that a translation is good.

Translation-pairs can be found in any bilingual dictionary, so a bilingual dictionary was used, all the while keeping in mind that the pairs are selected and the definitions are made in the lexicographical tradition and not finely tuned to frameworks and methods of machine-translation. There were no good ready-made tools to work with the available dictionary, so I had to make such tools first. The dictionary in question and the tools to deal with it are described in chapter 2, which also describes the other linguistic resources used in the remainder of the thesis. Several of these resources are made available through the larger project of which this thesis is a part, the LOGON project (Lønning et al., 2004). The first part of chapter 2 is therefore an overview of LOGON and of how and where this thesis fits in that whole.

The next chapter, chapter 3, dives into the theory of compounding and compound translation. Fortunately, I could draw upon existing studies of compounding and compound analysis in Norwegian. My primary source here has been a paper by Johannesen and Hauglin published in 1998. Earlier attempts at machine-translation of compounds are also discussed in this chapter: covering how to check for the validity of a translation by looking it up in a corpus of the target language (Rackow et al., 1992), solving the problem of limited size and coverage of a corpus by using the web as corpus directly (Grefenstette, 1999), the feasibility itself of using shallow translation of compounds (Tanaka and Baldwin, 2003a) and how to better select translation candidates and then how to evaluate them (Tanaka and Baldwin, 2003b).

While using the web as corpus is not a new idea, there have been, to my considerable and continuing surprise, very little to nothing properly published on how to do it right, compared to what we know of the use of deliberately collected corpora. There have been even less on how to utilize web search engines. Half of chapter 4 is therefore devoted to improving this situ-

ation. We then move on to the implementation of the compound translator `ReCompounder` in the second part of the chapter.

In order to be able to discuss search engine phenomena it has been necessary to refer to many blogs (web logs) and blogposts in chapter 4. These are not peer reviewed in the traditional sense, so I've used a limited selection. I've used two blogs run by linguists, namely *Language Log* which is run by the phonetician Mark Liberman, with guest writers including among others Geoffrey Pullum and Arnold Zwicky, and *Technologies du Langage* by the linguist and computer scientist Jean Véronis. Furthermore I've used the newsletters of *Search Engine Watch*, catering to search engine advertisers, and the *Yahoo! Search Blog*, run by the Yahoo! Search team. The final web-only source is for the quotation that starts chapter 4 itself. These sources all have an url in the bibliography.

Finishing discussions of the linguistic use of search engines we then turn to the compound translator itself. `ReCompounder` is a proof-of-concept prototype which tries to translate previously unseen compounds. It is part of the LOGON machine translation project which aims to translate tourism[4]-relevant texts from Norwegian to English, using an approach based on semantic transfer.

Thereafter follows concluding remarks in chapter 5.

As a service to lexicographers, words marked with the ♢ marks Norwegian compounds that lacks translations or mentions in the Norwegian-English dictionary[5] that was used for this project, like *testord*♢ 'test word'. There's a full list of these words in the index. Also note how all translations to English have been enclosed by ' and ', as in the example with 'test word' previously in this paragraph. Furthermore, all words, parts of words and terms used as examples are set in *italics* and have an entry in the index, sorted by language.

Anything that is to be typed on the command-line or given as queries to a search engine are in a `typewriter font`, names of computer programs will in addition have an entry in the index. When a term is first defined it is set

[4]More specifically hiking trip information

[5]Discussed in chapter 2.2 on page 9

in *italics*, with the corresponding entry in the index having its page-number set likewise.

In order to cut down on the amount of footnotes containing nothing but an url to more information, the urls of projects, programs and APIs have been moved to their own index, the *Homepages*-index on page 127.

Chapter 2

And now for something completely different

Monthy Python

The LOGON project and its linguistic resources

This chapter aims to give both a birds-eye view of the the machine translation project LOGON that this thesis was funded by and is part of, and to serve as a documentation of the linguistic resources available through LOGON that was used to build the `ReCompounder` as described in chapter 4.

We start with a description of the LOGON process and in particular its transfer-component, as it is here that a compound translation subcomponent will reside. Section 2.1.1 on page 7 will be of special interest to semanticists (and the users of HPSG and LFG) as it describes the semantic system used in the project, MRS, and how it connects to the other parts of the system.

What follows the section on LOGON is documentation for the linguistic resources used in this thesis *in addition to using the web as a corpus.* Section 2.2 on page 9 might be of special interest to the practically inclined as it documents a method for interacting with data encoded as XML that was discovered and implemented during the implementation-stage of the `ReCompounder`.

2.1 LOGON

LOGON (Lønning et al., 2004; Oepen et al., 2004) is an MT project that aims to translate text in the tourism-domain from Norwegian to English; more specifically, translating adverts for hiking-trips.

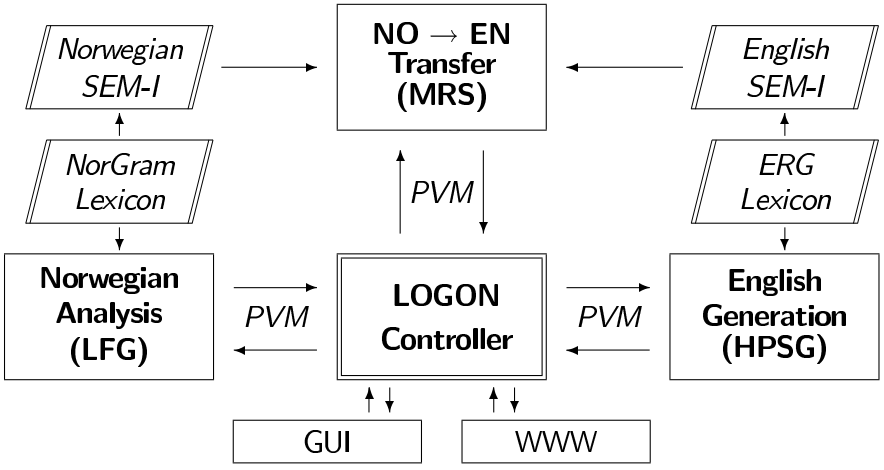


Figure 2.1: Schematic system architecture: the three core processing components (in boldface) are managed by a central controller that passes intermediate results (mrss) through the translation pipeline. The Parallel Virtual Machine (pvm) layer facilitates distribution, parallelization, failure detection, and roll-over. Figure from Oepen et al. (2004).

In the core system, Norwegian text analysis is handled by the LFG-based (Dalrymple, 2001; Bresnan, 2001) resource-grammar *NorGram* that is part of the *ParGram* (Butt et al., 1999) project via the XLE software from Xerox[1]. Generation of English text is facilitated by the HPSG-based (Pollard and Sag, 1994) LinGO English Resource Grammar (ERG) via additions to the LKB (Copestake, 2002) software system. The generating component is the brain-child of Carroll and Oepen (2005).

LOGON works by transferring Minimal Recursion Semantics (MRS) structures (Copestake et al., 1995), which aids in achieving one of the goals of the

[1]This is a gentle reminder that all addresses to important websites are collected in their own index on page 127 so as to reduce distracting visual clutter on the page.

project: translation that is as similar to the source language as semantically possible. In addition there'll be fallback methods to ensure that a translation will always result, even if it is not optimal. There's a cursory look at LOGON's MRS-dialect, how it is used with LFG and how Norwegian compounds are encoded in section 2.1.1.

Another purpose of LOGON is to create new and improve existing linguistic resources for Norwegian and Norwegian-to-English. The most important resource available for LOGON used in this thesis is the *Engelsk stor ordbok*, which is discussed in section 2.2 on page 9. Other resources available to LOGON are discussed in section 2.3 on page 31.

2.1.1 Minimal Recursion Semantics (MRS) in LOGON

While the project this thesis describes has not dealt directly with the MRS transfer-system, it is necessary to have a brief look at how MRS is used in the Norwegian part of LOGON. MRS has evolved from work on the semantics in HPSG, while the Norwegian grammar in LOGON is based on LFG. It has therefore been necessary to create a process for bridging the f-structures of LFG with the mrses[2] of HPSG (Dyvik et al., 2005).

Figure 2.2 on the following page shows the *c-structures*[3] as produced by XLE for two sentences only differing in their NPs. This particular structure was chosen to serve as an example due to its few words and lack of agreement[4], catering for small trees but complex *f-structures*.

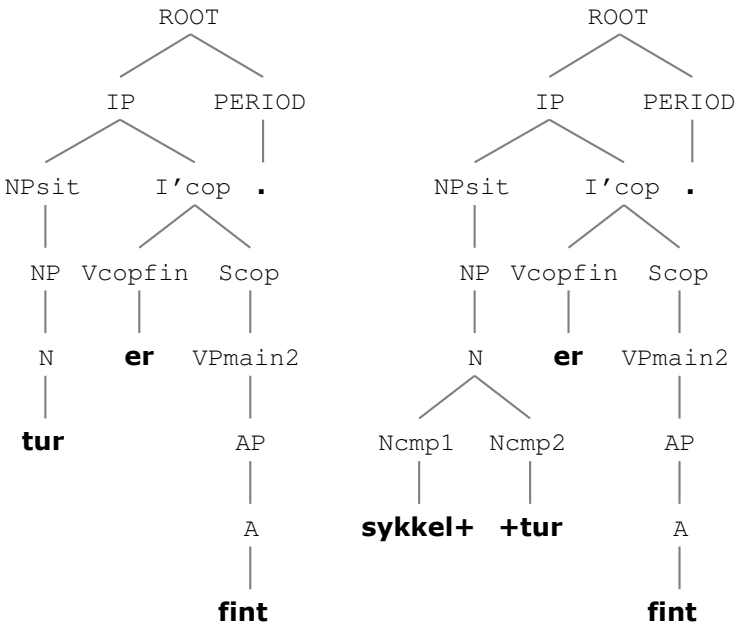
An *mrs* (shorthand for MRS structure) is projected from the f-structure of each sentence and then transferred to compatible mrses of the target language. The source language lemmas are transferred to their equivalents in the target language and any language-specific structures are added[5] or subtracted as necessary to construct the complete target language mrs. After the

[2]"MRS" is here used for the abstract notion while "mrs" is used for an actual MRS-representation of a specific sentence.

[3]The trees were made with the XLE-MRS demonstrator, see page on page 127.

[4]See Faarlund (1977), Corbett (1991, section 7.2.4) and especially Borthen (2003, chapter 9) for more about this phenomenon.

[5]E.g. *do*-support.



(a) "Tur er fint." (b) "Sykkeltur er fint."

Figure 2.2: Two example pseudo c-structures

mrses have been transferred, they are used as a basis to generate English sentences from, using techniques described in Carroll and Oepen (2005). This is outside the scope of this thesis and will therefore not be discussed any further.

2.1.1.1 Simple noun to mrs

The subject of the sentence in figure 2.2(a) is the simple noun *tur* 'trip, tour, hike'. The f-structure of figure 2.2(a) is in figure 2.3 on page 10, while the mrs projected from the f-structure is shown in figure 2.4 on page 11. The noun itself is to be found in the `OBJ`-node in the f-structure and the *_tur_n_rel* in the mrs.

During transfer, the symbols starting with an underline (_) in the mrs would be looked up in the lexicon and replaced with equivalent symbols from the ERG, e.g. *_sykkel_n_rel* would be replaced with *_bike_n_1_rel*.

The symbols not starting with an underline would be rewritten and replaced according to the transfer rules.

2.1.1.2 Compound noun to mrs

While figure 2.2(a) illustrated the use of a simple noun as the subject of a minimal sentence, figure 2.2(b) on the preceding page does that same for a compound noun, figure 2.5 on page 12 being its f-structure. The only difference between this sentence and the previous is that the simple noun *tur* have been replaced with the compound noun *sykkeltur* 'bike trip'. The mrses for the compound noun is shown in figure 2.6 on page 13.

2.2 The ENE dictionary by Kunnskapsforlaget

The *Engelsk stor ordbok* (Eek et al., 2001) (hereafter "*ENE dictionary*") is the most important linguistic resource used for the system described in this thesis. It is a large bilingual two-way dictionary of English and Norwegian. The Norwegian-to-English part contains 62554 word entries, of which 42635 are nouns, 8466 adjectives and 5027 verbs. There are also 472 "words" that only occur in expressions, like

(1) hulter til bulter

"pell-mell, helter-skelter, at sixes and sevens, in a mess"

where neither "hulter" nor "bulter" exists anywhere outside the expression.

The English-to-Norwegian entries have barely been used in this project.

Via LOGON, a digitized version of the ENE dictionary was available, encoded in XML[6]. The remainder of this section will describe how to use the XML-encoded dictionary, and dictionaries encoded in XML.

2.2.1 The structure of an entry

In this section we will start with a look at an abbreviated entry from the ENE dictionary, as shown in listing 2.1 on page 14.

[6]Well-formed but not valid XML as there is no DTD

$$
\begin{bmatrix}
\text{PRED} & \text{'være}\langle\text{[1:SIT-INVOLVES], [2:fin]}\rangle\text{'} \\
\text{TNS-ASP} & 3\begin{bmatrix}\text{TENSE} & \text{past} \\ \text{MOOD} & \text{indicative}\end{bmatrix} \\
\text{PREDLINK} & 2\begin{bmatrix}
\text{PRED} & \text{'fin'} \\
\text{GEND} & 6\begin{bmatrix}\text{NEUT} & + \\ \text{MASC} & - \\ \text{FEM} & -\end{bmatrix} \\
\text{NUM} & \text{sg} \\
\text{DEF} & - \\
\text{ATYPE} & \text{predicative}
\end{bmatrix} \\
0 & \begin{bmatrix}
\text{PRED} & \text{SIT-INVOLVES}\langle\text{5:tur}\rangle \\
\text{GEND} & \boxed{6} \\
\text{SUBJ} & 1\begin{bmatrix}
\text{PRED} & \text{'tur'} \\
\text{NTYPE} & 10\begin{bmatrix}\text{NSEM} & 12\begin{bmatrix}\text{COMMON} & \text{count}\end{bmatrix} \\ \text{NSYN} & \text{common}\end{bmatrix}
\end{bmatrix} \\
\text{OBJ} & 5\begin{bmatrix}
\text{GEND} & 9\begin{bmatrix}\text{NEUT} & - \\ \text{MASC} & + \\ \text{FEM} & -\end{bmatrix} \\
\text{PERS} & 3 \\
\text{NUM} & \text{sg} \\
\text{CASE} & \text{nom}
\end{bmatrix} \\
\text{NUM} & \text{sg}
\end{bmatrix} \\
\text{VTYPE} & \text{main} \\
\text{VFORM} & \text{fin} \\
\text{STMT-TYPE} & \text{decl} \\
\text{MAIN-CL} & +
\end{bmatrix}
$$

Figure 2.3: The f-structure of "Tur er fint.", notice especially the OBJ-avm and how the gender of *fint*, point 6 in the f-structure, clashes with the gender of *tur*, at point 9.

$$\begin{bmatrix}
\text{TOP} & \text{h1} \\
\text{INDEX} & \text{e4} \\
\text{RELS} & \left\{ \begin{bmatrix} \text{stative_asp_rel} \\ \text{LBL} \quad \text{h2} \\ \text{ARG} \quad \text{e4} \end{bmatrix}, \begin{bmatrix} \text{prpstn_m_rel} \\ \text{LBL} \quad \text{h1} \\ \text{ARG0} \quad \text{e4} \\ \text{MARG} \quad \text{h16} \end{bmatrix}, \begin{bmatrix} \text{udef_q_rel} \\ \text{LBL} \quad \text{h8} \\ \text{ARG0} \quad \text{x3} \\ \text{BODY} \quad \text{h9} \\ \text{RSTR} \quad \text{h7} \\ \text{LNK} \quad \text{2} \end{bmatrix}, \right. \\
 & \quad \begin{bmatrix} \text{sit-involving_rel} \\ \text{LBL} \quad \text{h10} \\ \text{ARG0} \quad \text{e11} \\ \text{ARG1} \quad \text{x6} \end{bmatrix}, \begin{bmatrix} \text{bare_sg_q_rel} \\ \text{LBL} \quad \text{h13} \\ \text{ARG0} \quad \text{x6} \\ \text{BODY} \quad \text{h14} \\ \text{RSTR} \quad \text{h12} \\ \text{LNK 0} \end{bmatrix}, \begin{bmatrix} \text{nominalize_rel} \\ \text{LBL} \quad \text{h15} \\ \text{ARG0} \quad \text{x3} \\ \text{ARG1} \quad \text{h10} \\ \text{LNK} \quad \text{2} \end{bmatrix}, \\
 & \quad \left. \begin{bmatrix} \text{_tur_n_rel} \\ \text{LBL} \quad \text{h5} \\ \text{ARG0} \quad \text{x6} \\ \text{LNK} \quad \text{0} \end{bmatrix}, \begin{bmatrix} \text{_fin_a_rel} \\ \text{LBL} \quad \text{h2} \\ \text{ARG0} \quad \text{e4} \\ \text{ARG1} \quad \text{x3} \\ \text{LNK} \quad \text{33} \end{bmatrix} \right\} \\
\text{HCONS} & \left\{ \text{h16 QEQ h2, h7 QEQ h15, h12 QEQ h5} \right\}
\end{bmatrix}$$

Figure 2.4: The mrs of "Tur er fint." Local MRS-dialect dictates that all actual lemmas are prefixed with _, as in *_tur_n_rel* for "tur". In addition to being connected through shared indices like *x6*, loose connections are made possible through the `HCONS`-list, for instance is *bare_sg_q_rel* loosely connected to *_tur_n_rel* via *h12 QEQ h5*, *h12* being *bare_sg_q_rel*'s `RSTR`.

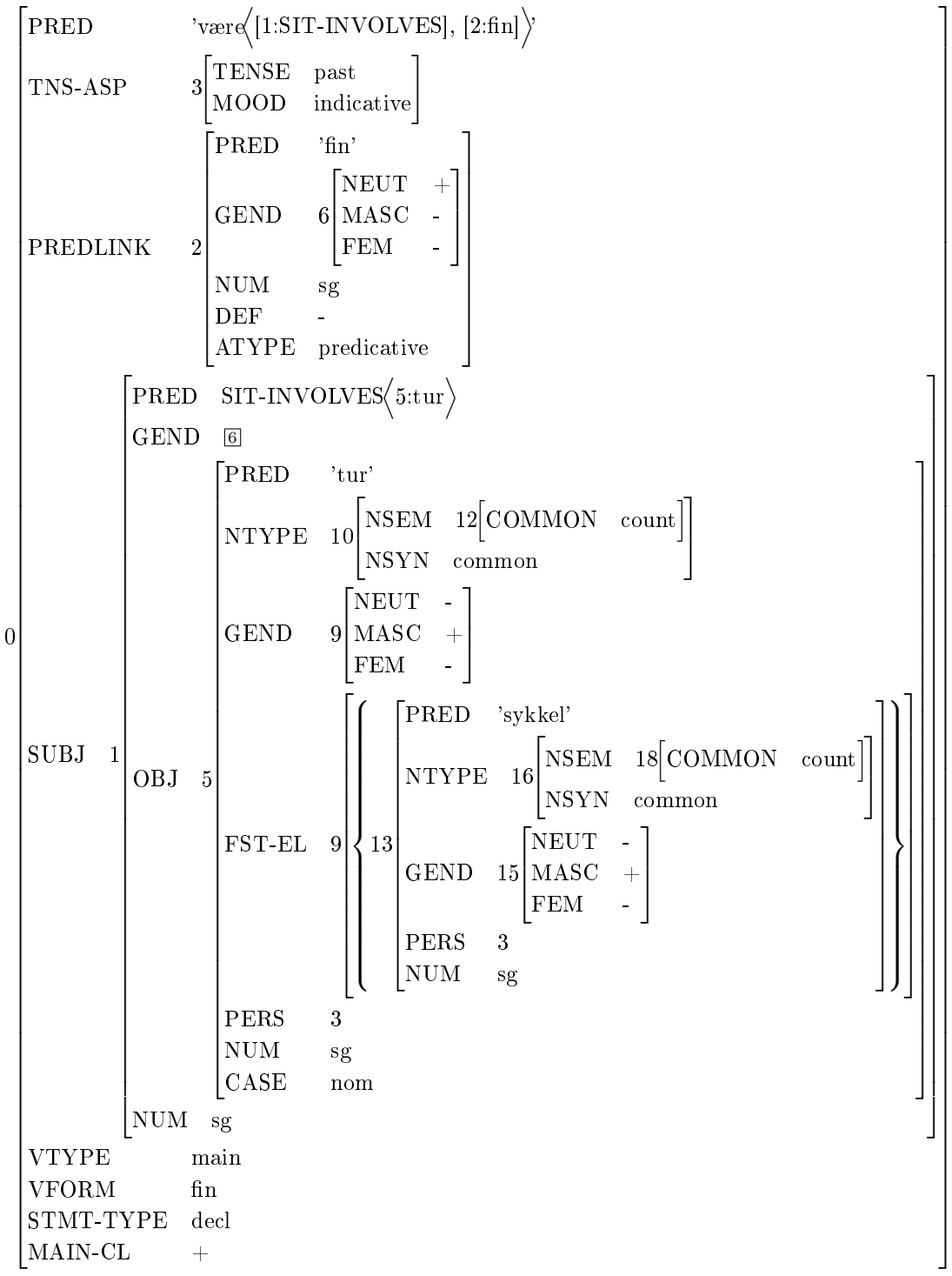


Figure 2.5: The f-structure of "Sykkeltur er fint." Compare the OBJ-avm here, a compound noun, with the non-compound noun OBJ-avm for "tur" in the previous f-structure.

$$\begin{bmatrix}
\text{TOP} & \text{h1} \\
\text{INDEX} & \text{e4} \\
\text{RELS} & \left\langle
\begin{bmatrix} \text{stative_asp_rel} \\ \text{LBL} \quad \text{h2} \\ \text{ARG} \quad \text{e4} \end{bmatrix},
\begin{bmatrix} \text{prpstn_m_rel} \\ \text{LBL} \quad \text{h1} \\ \text{ARG0} \quad \text{e4} \\ \text{MARG} \quad \text{h22} \end{bmatrix},
\begin{bmatrix} \text{udef_q_rel} \\ \text{LBL} \quad \text{h14} \\ \text{ARG0} \quad \text{x3} \\ \text{BODY} \quad \text{h15} \\ \text{RSTR} \quad \text{h13} \\ \text{LNK} \quad 6 \end{bmatrix}, \right. \\
& \quad \begin{bmatrix} \text{sit-involving_rel} \\ \text{LBL} \quad \text{h16} \\ \text{ARG0} \quad \text{e17} \\ \text{ARG1} \quad \text{x11} \end{bmatrix},
\begin{bmatrix} \text{bare_sg_q_rel} \\ \text{LBL} \quad \text{h19} \\ \text{ARG0} \quad \text{x11} \\ \text{BODY} \quad \text{h20} \\ \text{RSTR} \quad \text{h18} \\ \text{LNK} \ 16 \end{bmatrix},
\begin{bmatrix} \text{nominalize_rel} \\ \text{LBL} \quad \text{h21} \\ \text{ARG0} \quad \text{x3} \\ \text{ARG1} \quad \text{h16} \\ \text{LNK} \quad 6 \end{bmatrix}, \\
& \quad \begin{bmatrix} \text{unspec_rel} \\ \text{LBL} \quad \text{h10} \\ \text{ARG0} \quad \text{e12} \\ \text{ARG1} \quad \text{x11} \\ \text{ARG2} \quad \text{x8} \end{bmatrix},
\begin{bmatrix} \text{_tur_n_rel} \\ \text{LBL} \quad \text{h10} \\ \text{ARG0} \quad \text{x11} \\ \text{LNK} \quad 16 \end{bmatrix},
\begin{bmatrix} \text{udef_q_rel} \\ \text{LBL} \quad \text{h6} \\ \text{ARG0} \quad \text{x8} \\ \text{BODY} \quad \text{h7} \\ \text{RSTR} \quad \text{h5} \end{bmatrix}, \\
& \quad \left. \begin{bmatrix} \text{_sykkel_n_rel} \\ \text{LBL} \quad \text{h9} \\ \text{ARG0} \quad \text{x8} \\ \text{LNK} \quad 0 \end{bmatrix},
\begin{bmatrix} \text{_fin_a_rel} \\ \text{LBL} \quad \text{h2} \\ \text{ARG0} \quad \text{e4} \\ \text{ARG1} \quad \text{x3} \\ \text{LNK} \quad 30 \end{bmatrix} \right\rangle \\
\text{HCONS} & \left\langle \text{h22 QEQ h2, h13 QEQ h21, h18 QEQ h10, h5 QEQ h9} \right\rangle
\end{bmatrix}$$

Figure 2.6: The mrs of "Sykkeltur er fint." Notice how *_sykkel_n_rel* is connected to *_tur_n_rel* via *unspec_rel*: *unspec_rel* and *_tur_n_rel* sharing a `LBL` and *unspec_rel* linking to *_sykkel_n_rel* via the underspecified handle *x8* on the `ARG2`-list of *unspec_rel* and the `ARG0`-list of *_sykkel_n_rel*.

Listing 2.1: The entry for *hus* in the ENE dictionary. The entry has been indented for readability, and for the same reason all XML attributes have been removed and only two out of the 26 <uttrykkartikkel>-nodes remain. Missing line numbers indicate that the line is a continuation of the previous line.

```
1  <artikkel>
2    <hode>
3      <oppslagsord>hus</oppslagsord>
4      <ordkategori>
5        <ordklasse>subst.</ordklasse>
6        <kjoenn>n</kjoenn></ordkategori></hode>
7    <betydningsseksjon>
8      <betydning>
9        <veiviser>bygning</veiviser>
10       <oversettelse>
11         <ekv>house</ekv>,
12         <ekv>building</ekv></oversettelse></betydning>
13     <betydning>
14       <veiviser>hjem</veiviser>
15       <oversettelse>
16         <ekv>house</ekv></oversettelse></betydning>
17     <betydning>
18       <veiviser>husholdning</veiviser>
19       <oversettelse>
20         <ekv>house</ekv></oversettelse></betydning>
21     <betydning>
22       <veiviser>familie</veiviser>
23       <oversettelse>
24         <ekv>house(hold)</ekv>,
25         <ekv>family</ekv></oversettelse></betydning>
26     <betydning>
27       <veiviser>forretningsforetak</veiviser>
28       <oversettelse>
29         <ekv>house</ekv>,
30         <ekv>company</ekv>,
31         <ekv>firm</ekv></oversettelse></betydning>
32     <betydning>
33       <veiviser>kammer</veiviser>
```

```
34        <oversettelse>
35          <ekv>House</ekv>,
36          <ekv>Chamber</ekv></oversettelse>
37        <eksempelseksjon>
38          <eksempelpar>
39            <no-eksempel>det britiske parlamentet har to hus</
                 no-eksempel>
40            <eng-eksempel>the British Parliament has two Houses
                 </eng-eksempel>
41          </eksempelpar></eksempelseksjon></betydning>
42      <betydning>
43        <veiviser>hylster</veiviser>
44        <oversettelse>
45          <ekv>casing</ekv>,
46          <ekv>housing</ekv>,
47          <ekv>case</ekv></oversettelse></betydning></
               betydningsseksjon>
48    <uttrykksseksjon>
49      <uttrykkartikkel>
50        <uttrykk>trekke fulle hus</uttrykk>
51        <betydning>
52          <oversettelse>
53            <ekv>draw a crowded house</ekv>,
54            <ekv>draw crowds</ekv>,
55            <ekv>fill the houses</ekv></oversettelse>
56          <eksempelseksjon>
57            <eksempelpar>
58              <no-eksempel>Sting trakk fulle hus</no-eksempel>
59              <eng-eksempel>Sting drew crowded houses /
60                Sting filled the house</eng-eksempel>
61            </eksempelpar></eksempelseksjon></betydning></
                 uttrykkartikkel>
62      <uttrykkartikkel>
63        <uttrykk>v&#xE6;re herre i eget hus</uttrykk>
64        <betydning>
65          <oversettelse>se
66              <henvisning>herre</henvisning></oversettelse></
                   betydning>
67      </uttrykkartikkel></uttrykksseksjon></artikkel>
```

The root-node is <artikkel> starting on line 1, meaning *article*. Each entry then consists of a <hode>-node (line 2) containing the headword in the <oppslagsord>-node (line 3), and morphological paradigm-information on lines 5 to 6. Thereafter follows the <betydningsseksjon>-node (meaning-section, line 7), which contains one to several <betydning>-nodes, each with one or more translations of the headword. Instead of a translation the <betydning>-node can instead contain a <henvisning>-node, as on line 66, which means the rest of the information for that <betydning>-node is to be found in the entry given in the <henvisning>-node. The unreadable squiggle on line 63 is the Norwegian letter *æ* encoded in a safe way. Furthermore, each <betydning>-node can contain an optional <eksempelseksjon>-node, a section for examples, as shown on lines 56–61. Finally there might be an optional node, <uttrykksseksjon> (expression-section) as on lines 48ff, that contains one or more <uttrykk>-nodes, expressions containing the headword and an equivalent expression in the other language.

2.2.2 Efficient use of the ENE dictionary

LOGON has an SQL[7]-database version of the dictionary but it only includes the <hode>-nodes and <betydningsseksjon>-nodes without any of the expressions or examples. While it is possible to encode all the information of a tree-structure such as XML in one or more interconnected flat tables such as a relational database that can be queried with SQL, it is not a good solution since the very design of the relational database model came as a response to earlier, hierarchical models (Codd, 1970). Basically, neither the underlying database itself nor its query language are designed to contain information modeled as a tree-structure. It is *possible*, see for instance Celko (2004); Tropashko (2005), but as building a hierarchical system on top of a relational system is still considered state of the art I decided to find some other solution for this project.

In order to have access to the entirety of the tree it was therefore necessary to find and use tools intended for the task from the beginning. Furthermore,

[7]Structured Query Language

there were no known ready-made tools for quick manual lookups of words, expressions and examples at the time the project started. The solution chosen for this project is presented in the next section.

2.2.3 XML and how to use it

In order to explain any use of XML it is first necessary to clear up the jargon. Therefore I've made possibly the world's smallest XML-tutorial in listing 2.2 below.

Listing 2.2: The first line always declares the file to be XML. It must be present. The second line is a comment, identical to comments in HTML. These can go anywhere except inside a tag. The third line is the start-tag of the root-node, the tag <root>. There can only be one root-node per XML-file. Tags in XML may contain whitespace and one or more attribute-value pairs, as <root> does. Attribute-value pairs can always be rewritten to be daughters of the tag they are in. Open tags, as <root> or <openchild>, may enclose other tags and #PCDATA, which is all the text that is not a tag-name like <root>, an attribute-value pair or a comment. #PCDATA **must** be enclosed by tags and cannot go outside the root-node. A closed tag like <closedchild/> on line 6 cannot contain #PCDATA. All open tags must eventually be closed, as <root> is on line 7 with the end-tag </root>.

```
1 <?xml version="1.0"?>
2 <!-- I am a comment, I can go anywhere except inside tags -->
  <root attribute="value">
4   #PCDATA is <openchild>free text</openchild> that is neither
    a tag nor an attribute-value pair. #PCDATA must be enclosed
6   by tags. <closedchild/> is a tag that can't contain #PCDATA.
  </root>
```

XML is, in essence, just a way of encoding a tree[8], as figure 2.7 on the next page illustrates. Each XML-attribute and open XML-tag is a mother-node, each value and #PCDATA-token is a terminal node.

It is possible to use existing text-search tools like `grep` on XML, but these are generally unsuitable as they were designed to search character- or line-oriented data such as prose, source code or *flat*[9] text, while with XML

[8]With cross-references one can also encode more complex graphs.

[9]Data encoded in rows and columns, for instance comma-separated files.

the structure itself is supposed to be content-bearing information that needs to be searchable.

One way of extracting bits and pieces of XML directly is to use *XSLT* (Clark et al., 1999). However, XSLT is complex and verbose: it itself is encoded as XML and can be regarded as a notational variant of the programming language Scheme (Kelsey et al., 1998). Just as with Scheme, XSLT relies on functions and implicit recursion instead of syntax (in the form of operators and clearly delineated loops). Especially the length of function-names and lack of tools for interactive use makes it less useful for quick exploration from the command line[10], though it is well suited for batch processing.

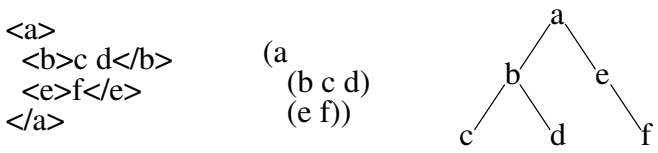


Figure 2.7: Equivalent trees: from left to right the same tree is encoded in XML, as s-expressions and visually.

One tool that makes the search in and extraction of trees possible is `tgrep2`, a follow-up to `tgrep` (tree-grep), that has for years been used to search the Penn treebank. `tgrep2` works on trees encoded as *s-expressions*, see e.g. the middle figure of figure 2.7, but not on trees encoded in XML. S-expressions differ from XML in that subtrees are enclosed by parentheses instead of tags and the first word/token after the opening parenthesis is the root of the subtree. Furthermore, s-expressions only have subtrees and hence no attributes.

SXML (Kiselyov, 2002) at `http://ssax.sourceforge.net` is a full reimplementation of XML as s-expressions. However, it seemed to be an overkill to use it in this project. Therefore it was decided to make a simpler alternative to just convert to and from generic XML.

[10]Its predecessor *DSSSL* has much the same drawbacks, and has largely been phased out.

2.2.3.1 Converting between XML and s-expressions

Conversion from XML to s-expressions was done with XSLT. Listing 2.3 shows the resulting generic stylesheet, which is small enough that adjustments "in the field" are possible.

Listing 2.3: Generic XSLT-stylesheet[12]to convert from XML to s-expressions. Line numbers are included as a convenience to the reader, and missing line numbers indicate that the line is a continuation if the previous line.

```
1  <?xml version="1.0"?>
2  <xsl:stylesheet
3      version="1.0"
4      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
5  <xsl:output method="text"/>
6  <xsl:strip-space elements="*"/>
7
8  <xsl:template match="text()">
9      (LEAF <xsl:value-of select="normalize-space(.)"/>)
10 </xsl:template>
11
12 <xsl:template match="*">
13     (<xsl:value-of select="local-name()"/>
14     <xsl:if test="@*">(@ <xsl:for-each select="@*">
15         (<xsl:value-of select="local-name(.)"/><xsl:text> </
               xsl:text>
16         <xsl:value-of select="."/>)</xsl:for-each>)</xsl:if>
17     <xsl:apply-templates/>)
18 </xsl:template>
19
20 </xsl:stylesheet>
```

It is necessary to run an XSLT-processor to use the stylesheet, and there are several available. Well-known open source alternatives include `Saxon` and `Xalan`, but I use `xsltproc` from the GNOME project. Simply running `xsltproc stylesheet xmlfile` will write the transformed xml to standard out, where it can be redirected to a file:

[12]Making this this stylesheet really made it clear to me how unsuitable XSLT is for quick explorations, taking several weeks longer than I had expected.

```
xsltproc stylesheet xmlfile > tgrep2ablefile
```

Running the stylesheet of listing 2.3 on the example ENE-entry of listing 2.1 on page 14 produces the s-expressions below, though without line numbers and indentation.

Listing 2.4: The converted entry for *hus* in the E-N-E dictionary. This has been made line-by-line equivalent to listing 2.1 on page 14 for ease of comparison.

```
1  (artikkel
2    (hode
       (oppslagsord (LEAF hus))
4      (ordkategori
         (ordklasse (LEAF subst.))
6        (kjoenn (LEAF n))))
     (betydningsseksjon
8      (betydning
         (veiviser (LEAF bygning))
10       (oversettelse
           (ekv (LEAF house)) (LEAF ,)
12         (ekv (LEAF building))))
       (betydning
14       (veiviser (LEAF hjem))
         (oversettelse
16         (ekv (LEAF house))))
       (betydning
18       (veiviser (LEAF husholdning))
         (oversettelse
20         (ekv (LEAF house))))
       (betydning
22       (veiviser (LEAF familie))
         (oversettelse
24         (ekv (LEAF house[hold])) (LEAF ,)
           (ekv (LEAF family))))
26     (betydning
         (veiviser (LEAF forretningsforetak))
28       (oversettelse
           (ekv (LEAF house)) (LEAF ,)
30         (ekv (LEAF company)) (LEAF ,)
```

```
        (ekv (LEAF firm))))
32    (betydning
        (veiviser (LEAF kammer))
34      (oversettelse
          (ekv (LEAF House)) (LEAF ,)
36        (ekv (LEAF Chamber)))
        (eksempelseksjon
38        (eksempelpar
            (no-eksempel (LEAF det britiske parlamentet har to hus))
40          (eng-eksempel (LEAF the British Parliament has two Houses)
          ))))
42    (betydning
        (veiviser (LEAF hylster))
44      (oversettelse
          (ekv (LEAF casing)) (LEAF ,)
46        (ekv (LEAF housing)) (LEAF ,)
          (ekv (LEAF case)))))
48  (uttrykksseksjon
      (uttrykkartikkel
50      (uttrykk (LEAF trekke fulle hus))
        (betydning
52        (oversettelse
            (ekv (LEAF draw a crowded house)) (LEAF ,)
54          (ekv (LEAF draw crowds)) (LEAF ,)
            (ekv (LEAF fill the houses)))
56        (eksempelseksjon
            (eksempelpar
58            (no-eksempel (LEAF Sting trakk fulle hus))
              (eng-eksempel (LEAF Sting drew crowded houses /
60              Sting filled the house))
              ))))
62    (uttrykkartikkel
        (uttrykk (LEAF være herre i eget hus))
64      (betydning
          (oversettelse (LEAF se)
66            (henvisning (LEAF herre))))
    )))
```

It should be noted that the stylesheet of listing 2.3 on page 19 convert *absolutely everything* in the XML-file, even nodes and attributes that are not

necessary. Since `tgrep2` operates on a *forest*[13] of trees and not on a super-tree, it may be necessary to remove several layers of top-nodes in order to concentrate on the entries themselves. How to do this is detailed in the next numbered section.

How to convert from s-expressions to XML follows for completeness' sake. Both SXML as described on page 18 and the program provided in appendix A.2.2 can be used for this.

The latter is a standalone Python program that I made to serve this function, implemented by a simple finite state transducer with the addition of a stack for the xml-tags. The source for this is included on page 109 onwards. It is a generic s-expressions-to-XML-converter and does not treat subtrees whose roots are `@` or `LEAF` in any special way.

2.2.3.2 Tuning the stylesheet

To help illustrate usage and tips, the XML-encoded dictionary in listing 2.5 will be used. This is an abstraction of the format of the *Engelsk stor ordbok*, with all node-names in English.

Appendix A contains the stylesheet (listing A.2 on page 107) that results after adapting the generic stylesheet of listing 2.3 to the simulated dictionary. This shows how to strip away attributes and the mother-node. Compare listing A.1, which was converted with the generic stylesheet described before, to listing A.3, using the adapted stylesheet.

Listing 2.5: Example of a hypothetical dictionary of English, XML-encoded.

```
<?xml version="1.0"?>
<dictionary>
  <entry number="1">
    <word>abacus</word>
    <pos grammarpage="nouns">n</pos>
    <definition>ancient manual calculator</definition>
    <seealso>slide rule</seealso>
```

[13] Alternatively: there is no abstract mother-node *collection-of-entries*, only sibling-trees of *entries*.

```
8     </entry>
9     <entry number="2">
10      <word>abash</word>
        <pos grammarpage="verbs">vt</pos>
12      <definition>to make so. ashamed or embarrased</definition>
        <expression>
14        <words>to abash so. by sneering</words>
        </expression>
16    </entry>
      <entry number="3">
18      <word>abeyance</word>
        <pos grammarpage="expressions">n</pos>
20      <definition>in expressions only</definition>
        <expression>
22        <words>(property that is) in abeyance</words>
          <meaning>property without an owner</meaning>
24      </expression>
        <expression>
26        <words>hold smth. in abeyance</words>
          <meaning>temporarily suspend smth.</meaning>
28      </expression>
      </entry>
30  </dictionary>
```

Being a proper XML-file, the example starts with the XML-header on line 1. The root-node, <`dictionary`> on line 2, contains one <`entry`>-node per headword (line 3). Nodes of the made-up example can also carry attributes, specifically a unique id (`number`, line 3) or a link to an external resource (`grammarpage`, line 5).

Converting example 2.5 with the generic stylesheet in listing 2.3 on page 19 won't produce anything of use to `tgrep2` however. Listing 2.6 on the following page shows the first seven lines (the entirety of the converted file is to be found in listing A.1 on page 105 in the first appendix).

Take note of the following: there is a single root node, and `tgrep2` can only work with root-nodes of a limited size. If if we were dealing with a thousand entries or more instead of just three, `tgrep2` would have to give up. It is therefore prudent to remove the root-node of the example, turning

Listing 2.6: The first seven lines resulting from converting the example XML without adapting the stylesheet.

```
1 (dictionary
2
    (entry
4     (@
        (number 1))
6     (word
        (LEAF abacus))
```

it into a forest of <entry>-trees. Furthermore, studying the original data reveals that the information carried by the number-attribute is redundant (seemingly encoding the order of the entries), so it is preferable to strip away these as well.

Listing 2.7: Parentheses in the XML leads to extraneous subtrees. Lines 44–47 of listing A.1 on page 105, showing an extra subtree for the word "property" on line 46.

```
44        (words
            (LEAF
46            (property that is)
              in abeyance))
```

In lines 44 to 47 of the same example we find another problem: the fragment "property that is"[14], which was enclosed in parentheses in the XML, have become a subtree of its own. This might needlessly complicate searches. For example, normally terminal-nodes can be extracted by using the -t argument to tgrep2, this will now miss "property" since it has become the mother of "that is". Furthermore, converting back to XML using a naive converter[15] will produce <LEAF><property>that is</property>in abeyance</LEAF> instead of <LEAF>(property that is) in abeyance</LEAF>.

The rest of this section will tune the generic stylesheet to the XML at

[14] tgrep2 will treat any single word enclosed in parentheses as if it was a terminal node.

[15] For instance a converter not utilizing a DTD (document type definition) but directly turning any word after an opening parenthesis into an XML-node.

hand, resulting in the s-expressions given in its entirety in listing A.3 in the first appendix.

Preparing XML for tgrep2 As shown in listing 2.7 on the facing page, `tgrep2` *will* consider all opening parentheses to branch off a new subtree. If the source XML contains text enclosed in parentheses that should be treated as regular text, it becomes necessary to remove or replace all parentheses in the XML by e.g. square brackets, for instance by using the standard search&replace-command of a text-editor.

Line 19 in the example XML in listing 2.5 is a case in point, compare lines 45 and 46 in listing 2.7 with line 31 in listing 2.8:

Listing 2.8: The situation in listing 2.7 is avoided by replacing the surplus parentheses in the source XML with square brackets prior to conversion, preventing the formation of a subtree.

```
30      (words
          (LEAF [property that is] in abeyance))
```

Selecting the trees we want to convert An XML-file generally includes only a single tree and not several (aka. a forest), and therefore just a single root-node. In example 2.5, that node is <`dictionary`>.

Now, we won't need this <`dictionary`>-node in any of our `tgrep2`-trees because we will concentrate on the <`entry`>-nodes. By adjusting line 12 in the XSLT from

```
12 <xsl:template match="*">
```

to

```
12 <xsl:template match="dictionary//*">
```

only the descendants of <`dictionary`> but not <`dictionary`> itself is included in our `tgrep2`-able trees. Compare the first three lines of listing 2.6 on the facing page with the first line of listing A.3 on page 107, reproduced below, where the root-node has been removed.

Listing 2.9: After adapting the stylesheet, listing 2.6 on page 24 now looks like the five lines below.

```
1 (entry
2   (@
      (number 1))
4   (word
      (LEAF abacus))
```

Ignoring attributes None of the attributes of the example are of use so they will be stripped from the trees.

This can be accomplished by deleting[16] the lines 13 to 16 inclusive from listing 2.3 and no attributes will be used in the trees. Otherwise, the attributes of a node will be the first child subtree of the mother node, see figure 2.8.

```
<node attr1="1" attr2="2" />  ⟺  (node (@ (attr1 1) (attr2 2)))
```

Figure 2.8: A node with attributes in XML and its equivalent s-expression

To see the effect of doing this, compare listing 2.6 with the start of listing A.3 on page 107, reproduced below, where the root-node and all of the attributes have been stripped away.

Listing 2.10: The result of removing both the root-node and attributes from listing 2.6 on page 24.

```
1 (entry
2   (word
      (LEAF abacus))
```

What to do with free text Free text, or #PCDATA in XML-jargon, is content without explicit XML structure; text-strings that are neither a tag nor an attribute. XML-relatives like HTML or MathML varies in how much and where they allow #PCDATA, if any.

[16] Alternatively: comment out by prepending <!− − and postpending −− >

Since a #PCDATA node is always a terminal node, I decided to make them structurally different from non-#PCDATA terminal nodes[17]. I did this by turning them into sequential daughters of the abstract mother-node `LEAF`.

By changing line 9 of listing 2.3 from

```
9   (LEAF <xsl:value-of select="normalize-space(.)"/>)
```

to

```
9   <xsl:value-of select="normalize-space(.)"/>
```

the #PCDATA become daughters of their logical mother node instead. Compare line 4 in listing A.1 on page 105 and lines 6 and 7 in listing A.3 on page 107 with figure 2.7 on page 18, which does not use `LEAF`.

2.2.3.3 Using `tgrep2`

Several examples of how to use `tgrep2` as a dictionary lookup-tool follows. The database used to illustrate every example is the mini-dictionary in listing 2.5 on page 22.

Making the `tgrep2`-database It is necessary to compile a database for `tgrep2` from the s-expressions. Do this by using the `-p <s-expression file>` argument:

```
tgrep2 -p tgrep2ablefile tgrep2database.t2c
```

`tgrep2ablefile` is the filename of a file containing the forest of s-expression trees while `tgrep2database.t2c` is the name that will be given to the resulting `tgrep2` database.

Consult the documentation for `tgrep2` for more options.

Showing all entries The following example shows how to list all the entries in the dictionary. The query must always be in single quotes, and the leftmost node in the query will be the mother-node of the returned

[17]The way the ENE dictionary is structured, a non-#PCDATA terminal node seems to be an error, but this is not necessarily the case for other dictionaries encoded in XML.

trees, if any. The corpus-file to use can be given with either the command-line argument `-c <tgrep2-database file>`, or the environment variable TGREP2_CORPUS. In this example we give the name of the database directly.

```
tgrep2 -c tgrep2database.t2c 'entry'
```

The results are identical to listing A.3 on page 107 apart from the line-numbers and indentation.

Listing all possible part-of-speech tags used For this and the remaining examples, the full path to the database (the mini-dictionary of listing 2.5 on page 22) have been set in the TGREP2_CORPUS environment variable prior to execution of the command.

```
tgrep2 'pos' | sort -u
```

```
(pos (LEAF n))
(pos (LEAF vt))
```

Just mentioning the pos-tag will list all pos-tags in the entire database, so the output is sent through a filter that sorts the lines alphabetically and removes duplicates.

Listing only trees that describe transitive verbs This uses both the descendant-operator << and the grouping-parentheses. The first tag denotes which node the subtrees should have as root while all other operators and tags select which specific subtrees to show.

```
tgrep2 'entry << (pos << vt)'
```

```
(entry (word (LEAF abash))
  (pos (LEAF vt)) (definition (LEAF to make so. ashamed or embarrassed))
  (expression (words (LEAF to abash so. by sneering))))
```

Searches like this of course depends on what data is included in the dictionary. For instance, the ENE dictionary doesn't mark its verbs for transitivity in any way.

Listing only trees whose headwords end with "sh" You can use regular expressions to search within a node-name, enclosed by two slashes:

```
tgrep2 'word << /sh$/'
```

```
(word (LEAF abash))
```

Listing only trees that contain a particular phrase The `$`-operator indicates that the words or tags it binds are sisters, and can be used to select trees containing a specific phrase.

```
tgrep2 'entry << (property $ in $ abeyance)'
```

```
(entry (word (LEAF abeyance))
  (pos (LEAF n))
  (definition (LEAF in expressions only))
  (expression
    (words (LEAF property in abeyance))
    (meaning (LEAF property without an owner)))
  (expression
    (words (LEAF hold smth. in abeyance))
    (meaning (LEAF temporarily suspend smth.))))
```

2.2.3.4 Translation by `tgrep2`

It is possible to use `tgrep2` to make a crude, word-to-word baseline translation of a text given that the words of the source-text have been lemmatized. This section will be drawing examples from the full *Engelsk stor ordbok* XML database.

The right pattern with `tgrep2` and the ENE dictionary can provide translation-suggestions. With more complex patterns it is possible to use the suggestions directly, as they stand it is necessary to remove the `ekv`-node and the `LEAF`-node.

Words: what is an "abbed"?

```
tgrep2 'ekv >> (artikkel << abbed)'
```

```
(ekv (LEAF abbot))
```

The descendant-operator >> is new here.

Expressions: what does "hulter til bulter" mean?

```
tgrep2 -a 'ekv >> (artikkel << (uttrykk << (hulter $ til $ bulter))'
```

```
(ekv (LEAF pell-mell))
(ekv (LEAF helter-skelter))
(ekv (LEAF at sixes and sevens))
(ekv (LEAF in a mess))
```

Running the first pattern on each word of a text yields a word-for-word translation (example 2 below) while the second can be used if collocations have been identified in the source text[18].

(2) a. Manual translation:

fotball er kjedelig
fotball være kjedelig

'Soccer is boring'

b. Translations of each lemmatized word, in order found:

i. *fotball*: 'football', 'soccer', 'Association Football', ...

ii. *være*: 'be', 'exist', 'wait', ...

iii. *kjedelig*: 'dull', 'boring', 'tedious', ...

c. Word by word translation using first found translation:
'football be dull'

[18] `tgrep2` *does* care about punctuation, in stark contrast to web search engines, as lamented in chapter 4.

2.3 Other linguistic resources available through LOGON

This section discusses the other linguistic resources that was used to build the compound translator, apart from the use of the web as a linguistic resource, which will be discussed in chapter 4 on page 57.

2.3.1 NorKompLeks

The NorKompLeks project (NKL) (Nordgård, 1996) aimed to produce machine-readable computational linguistics oriented dictionaries and wordlists for the two varieties of written Norwegian, Bokmål and Nynorsk. NKL consists of lists of basic and inflected form of all words in Bokmålsordboka and Nynorskordboka and morphological information for the inflected forms. It also contains a rendering of the pronunciation for each word and valency-information for the verbs.

In LOGON the NKL-lists find many uses. For instance, instead of lemmatizing an inflected form, it is often sufficient and quicker to look it up in the correct list, sometimes avoiding a potentially ambiguous algorithm-based morphological analysis in the process. Furthermore it acts as a check on and source for vocabulary coverage. Finally, the forms as recorded in NKL are currently being compared to forms as found in the wild by the LOGON lexicon-team; that a word *can* be inflected doesn't mean that it ever *is*. Discovering such discrepancies will cut down on ambiguity since the words in question can be marked as exceptions, thus blocking some morphological analyses.

Listing 2.11: The noun *ligning*, with spelling variants, as found in NorKompLeks. The "→" denotes a tab-character in the original data.

```
1 ligning→ num=sg form=ind gend=f pos=subst status=nf→ligning
2 ligning→ num=sg form=ind gend=m pos=subst status=nf→ligning
  ligninga→ num=sg form=def gend=f pos=subst status=nf→ligning
4 ligningen→ num=sg form=def gend=m pos=subst status=nf→ligning
  ligningene→ num=pl form=def gend=f pos=subst status=nf→ligning
6 ligningene→ num=pl form=def gend=m pos=subst status=nf→ligning
```

```
ligninger ⟶ num=pl form=ind gend=f pos=subst status=nf↦ligning
ligninger ⟶ num=pl form=ind gend=m pos=subst status=nf↦ligning
likning↦ num=sg form=ind gend=f pos=subst status=nf↦likning
likning↦ num=sg form=ind gend=m pos=subst status=nf↦likning
likninga ⟶ num=sg form=def gend=f pos=subst status=nf↦likning
likningen ⟶ num=sg form=def gend=m pos=subst status=nf↦likning
likningene ⟶ num=pl form=def gend=f pos=subst status=nf↦likning
likningene ⟶ num=pl form=def gend=m pos=subst status=nf↦likning
likninger ⟶ num=pl form=ind gend=f pos=subst status=nf↦likning
likninger ⟶ num=pl form=ind gend=m pos=subst status=nf↦likning
```

Listing 2.11 on the preceding page shows how the word form "ligning", meaning *tax assessment* or *equation*, is encoded in NorKompLeks. It has the spelling variant "likning" on lines 9 to 16, and inflects both as masculine/-common and feminine, as seen in lines 3 and 4. "Ligning" partakes in many compounds, as listed in table 2.1.

Norwegian compound	English translation
prosentligning	imputed rental value/income
skatteligning	tax assessment
skattligning	tax assessment
skjønnsligning	estimated tax
andregradsligning	quadratic equation
annengradsligning	quadratic equation
differensialligning	differential equation
eksponentialligning♢[19]	exponential equation
førstegradsligning	linear equation
tilstandsligning♢	equation of state
tredjegradsligning	cubic equation

Table 2.1: A subset of the nouns ending with "ligning" in NorKompLeks. The first four words are used in the tax domain, while the rest are to be found in the natural sciences.

[19] A quick reminder: the diamond (♢) marks all Norwegian words lacking entries in the ENE dictionary. A complete list of these are to be found in the index.

2.3.2 The evaluation corpus

This is a small corpus built by Ola Huseth in January 2004. A subset of it was used[20] as a source of compounds for evaluating the compound translator `ReCompounder` described in section 4.2 on page 79. The subset consists of 114 "sentences", where a sentence is defined as "line that ends with sentence-ending punctuation".

The corpus is a collection of tourist information built from web pages that had both a Norwegian and an English version. Stylistically they resemble advertisements (example 3, my translations), having a high frequency of adjectives, intensifiers and proper nouns per sentence, often speaking directly to the reader, as in example 3d.

(3) Typical sentences

a. Molde og Åndalsnes har et rikt utvalg av spisesteder, barer, puber og diskotek med dansetilbud.
'Molde and Åndalsnes are well supplied with restaurants, bars, pubs and clubs with dance floors.'

b. Ute ved kysten ligger de idylliske fiskeværene Ona, Bjørnsund, Bud og Håholmen.
'Out by the coast you'll find the idyllic fishing villages of Ona, Bjørnsund, Bud and Håholmen.'

c. Fotturen fra parkeringsplassen og ut til selve Preikestolplatået er bare 4 kilometer, men tar likevel ca 2 timer hver vei.
'The hike from the parking lot and to the Preikestol plateau itself is only 4 kilometers, but it still takes about 2 hours each way.'

d. På Fosen kan du bo i flotte rorbuer og sjøhus like ved sjøkanten.
'At Fosen you can stay in great cabins and houses just by the edge of the ocean.'

Several of the sentences are *fragments*, like example 4a on the following page that lacks a verb, or lists like in example 4b on the next page.

[20]It has also been used by Johansen (2006) for developing and evaluating automatized evaluation techniques like BLEU (Papineni et al., 2002).

(4) Fragments

a. Bra skiltet.

b. Helleristninger, huler, runer, gravrøyser og bautasteiner.

It also contains spelling errors, non-standard spellings like **souvenir* for *suvenir* and dialectal terms like *boss* for *søppel*, 'garbage'.

Since it is used as a reference-corpus for evaluation and to test evaluation-methodology, several manual translations to English of every sentence are provided. This project only used the Norwegian originals.

2.3.3 The tourist-corpus

The tourist training corpus is a collection of bitexts on hiking by foot in Norway, the domain of LOGON. It is documented in depth on its own web page at `http://www.hf.uio.no/tekstlab/prosjekter/tourist/`.

The text is collected from tourist websites in Norway, and some of the properties of the domain can be seen in example 5. The word-by-word translations are provided by the author while the free translations are from the corpus itself. The texts are meant to entice tourists to go hiking so adjectives and intensifiers like "marvelous", "fantastic" and "wonderful" are frequent. The example also demonstrates the variable quality of the translations, 5a being fairly good while 5b[21] is bad to the point of being funny.

(5) a. Her får du en fantastisk utsikt !
here get you a fantastic view
"You will have a marvellous view!"

b. Mørke skyer i skiftende kontrast med blank og grå himmel
dark clouds in shifting contrast with clear and grey sky
er aktørene i et fantastisk skuespill hvor selve
is the.actors in a fantastic play where itself
Atlanterhavet fungerer som scene .
the.Atlantic.ocean functions as stage

[21] This is an interesting sentence for machine translation. Even though it is fairly long, the real challenges are due to the ambiguity of *blank* and *selve*, the latter capable of being an adverb and taking part in several more or less fixed expressions.

"Spectacular in winter."

I've used this corpus to familiarize myself with the domain LOGON is to translate, and to see how the main LOGON-project was doing.

2.3.4 Norsk ordbank

A wordlist from Norsk ordbank was used to increase the number of stems available. *Norsk ordbank* is a badly documented morphological database built up from Bokmålsordboka (Wangensteen, 2005), Nynorskordboka (Hovdenak, 2001), word-lists collected by IBM Norge A/S and verb data from NorKomp-Leks (Nordgård, 1996). Norsk ordbank is built and hosted by the Unit for Digital Documentation at the Faculty of Arts, university of Oslo.

2.3.5 Unnamed resources

In addition to the linguistic resources mentioned above there are some smaller, unnamed frequency lists and POS-tagged texts in use, often being subsets of resources mentioned earlier in this chapter.

Chapter 3

"You shall know a word by the company it keeps."

(Firth, 1957)

Compounds and other oddities

Since the goal of this project (and hence the subject matter of this thesis) is to look at ways to automatically translate compounds it is prudent to discuss both how "compounds" are defined and used in this thesis and how they are defined and used elsewhere. This is done in the first section of this chapter. Afterwards follows a section on compounds in Norwegian and their equivalent constructions in English. The third section concentrates on how to translate compounds while the last section briefly looks into phenomena that may be translated using similar techniques to translating compounds.

3.1 Compounds and compounding in general

A *compound* is one of the many terms in linguistics that needs to be defined per language, as is also the case for tense, aspect, mood and other morphosyntactic terms. A generic definition as "two or more words that stand in a special relationship to each other in the same phrase and become a new word, possibly with a meaning derived from the component words" is a bit too vague to work with. In Norwegian and German for instance, a compound is always one or more stems that are orthographically written[1] without a single intervening space, but they can also be conjoined with a hyphen. In English the orthographic picture is not as clear cut, as Laurie

[1]Since stress is not marked in writing I've conveniently chosen to completely ignore it.

Bauer recently pointed out (Bauer, 2004)[2].

Furthermore, when using a resource produced by lexicographical principles like the ENE dictionary, it becomes necessary to look at the lexicographical definitions of terms in addition to the linguistic ones. Fortunately there is a dictionary of such terms, the *Nordisk leksikografisk ordbok* by Bergenholtz et al. (1997), which contains corresponding terms of lexicography in seven languages: Danish, English, Finnish, French, German, Icelandic, Norwegian (bokmål), Norwegian (nynorsk), and Swedish. The definitions themselves are in Norwegian (bokmål) in the referenced copy.

In this dictionary, a compound[3] is defined as a word consisting of two or more core morphemes[4], or word-bases, each which might itself be compound. It is also noted that the reasons to include a compound in a dictionary is not limited to recording its meaning, but also to record its spelling (for instance with or without hyphen or space) and morphological changes if different from the morphology of the component stems in the compound (my translation).

There is a question whether a compound is only those combinations of words whose meaning can be guessed from its constituent words alone, that it must be *transparent*, *analyzable* or fully semantically compositional[5]. This then might exclude "compounds" that are not transparent, or unanalyzable, for instance the subtype of compounds known as *bahuvrihi*, itself an example of bahuvrihi, as it is Sanskrit for 'much rice' meaning a rich person. An English example would be *redhead*, which is a person having red hair, and an example from Norwegian is *brunskjorte* 'brown shirt', which is another term for a Nazi (Nazis wore brown shirts). Bahuvrihi compounds never contain their own heads and can thus be tricky to translate. Another type of unanalyzable compound are words like English *blackbird*, which is particular

[2]Two words written without intervening spaces doesn't necessarily have compound stress, two words written with intervening spaces can have compound stress.

[3]Norw. *sammensetning*

[4]Core morphemes: Morphemes carrying meaning that can stand alone, as opposed to inflections and derivations, even though Bergenholtz et al. (1997) defines it only as a "necessary part" of any noun, verb or adjective.

[5]Notice that none of the above definitions mention that the semantics of the compound must be compositional.

species of bird *(turdus merula)* that happens to have black plumage[6] and not any bird that happens to appear black. Since the head is included in this type, translating stem by stem will yield a translation that gives an idea of the meaning of the word as a whole but not necessarily an exact translation. In the case of *turdus merula*, the Norwegian word is *svarttrost*.

In English, compounds (like "iced tea") often consist of several orthographic words, while in Norwegian and German they always consist of only one orthographic word. This leads to a potential conflict of definitions: a compound in Norwegian remains a compound even if fully lexicalized, while a compound in English may be seen as lexicalized and therefore switch categories from compound to word as soon as its constituent words are no longer written separated by spaces. Furthermore nouns preceding nouns in the same noun phrase in English can also be a case of what Huddleston and Pullum (2002, p. 453) calls *residual pre-head modifiers*. Examples include *passenger aircraft* and *winter's day*, both of which are compounds (*passasjerfly*[7] and *vinterdag* respectively) in Norwegian. There are guidelines for discovering whether a noun+noun or adjective+noun construction are compound or *composite nominals* on page p448f in Huddleston and Pullum (2002).

Finally, as compounds in English are considered to be multiword expressions (MWEs) by some, it is necessary to look at and define the term.

The lexicographic dictionary defines three relevant terms:

multiword lexical units[8] are common in languages which write the stems of compounds separately like English (*Member of Parliament*) and French (*pomme de terre*), according to Zgusta's 1971 criteria:

1. change of word order is impossible (**Parliament of Member*)
2. often impossible to separate the words in the unit by other words (**pomme de bonne terre*)

[6]For some, "Blackbird" primarily means the SR-71 Blackbird, a now retired American supersonic reconnaissance jet plane.

[7]This word has a translation error in the ENE dictionary, having been translated to 'passenger airline' in addition to 'passenger plane'.

[8]Norw. *flerordsenhet*

3. non-compositional semantics (*pomme de terre* is not an 'apple in/of the earth')
4. one or more words in the unit has a meaning only possible in the particular unit (an *old maid* need not be either old or a maid)
5. the unit often has a one-word synonym (*kick the bucket* vs. *die*)

Multiword lexical units include both fixed expressions, citations, proverbs and set phrases, and less fixed structures like collocations.

multiword lemmas[9] are the subset of multiword lexical units that have the same status as a single word, as for instance French *pomme de terre*, and therefore needs standalone entries in a dictionary.

multiple-word terms[10] are multiword lexical units specific to a profession or technical field.

The linguistic tradition doesn't seem to have settled for a term, and out of the partly to fully overlapping contenders (*multiword unit*, multiword expression, *collocation*, *Non-Compositional Compound* (*NCC*) (Melamed, 1997) etc.) this thesis will use *multiword expression* (*MWE*) (Sag et al., 2002), in line with the *MWE-project* at Stanford, whose working definition is *any phrase that is not entirely predictable on the basis of standard grammar rules and lexical entries.* The qualities and types of MWEs as listed in `http://mwe.stanford.edu/reading-group.html` overlaps with the lexicographical definition of multiword lexical units above.

The MWE project uses Moon's categories (Moon, 1998) which don't mention compounds outright, but the MWE-project speculates that *any expression that can be realized hyphenated/as a single lexeme or alternatively with spaces (e.g. mailman/postman vs. mail man/post man), is an MWE*, and *postman* is definitely a compound. Ergo, an English compound, at least writable with spaces or hyphens, is a multiword expression.

[9] Norw. *flerordslemma*

[10] Norw. *flerordsterm*

The question to ask is then: If English compounds are MWEs, are transparent translations into German or Norwegian of those compounds also multiword expressions, even when proper orthography dictates that spaces and hyphens are not to occur?

Since the exact definition of compounds and MWEs in English and other languages is a subject worthy of a thesis on its own, this thesis will not debate the question any further, especially when or if a word form ceases to be a compound and instead turns into a proper lexical item. Instead a very pragmatic stance is taken: when it is *useful* to treat something as if it was a compound, it is treated as if it *is* a compound, whether lexicalized or not.

3.2 Compounds and compounding in specific

Compounding[11] as a word-formation process is *very* productive in Norwegian. Munthe (1972) could report that 10.4% of words in running text were compounds. Thus any MT-system translating to or from Norwegian needs a way to handle newly minted compounds that were not found in the system's lexicon.

The LOGON-project, as described in the previous chapter, has access to several dictionaries and wordlists. The most important for this project was the ENE dictionary, as described in section 2.2 on page 9.

3.2.1 Compounds in Norwegian

The list of examples marked 6a) to e) below starts with a compound that were not found in any of the available dictionaries and wordlists, both text-versions and machine-readable, and derives other compounds from it:

(6) a. *gårdshus*◇
gård+hus
'farmhouse'

[11]To improve the coverage of compounds, HPSG has recently acquired a *compound matrix* (Søgaard, 2004).

b. *gårdshustak*♢
gård+hus+tak
'farmhouse roof'

c. *gårdshustakstein*♢
gård+hus+tak+stein
'farmhouse roof tile'

d. *gårdshustaksteinsproblem*♢
gård+hus+tak+stein+problem
'problem with the roof tiles of farmhouses'

e. *storgårdshus*♢
stor+gård+hus
'house of a large farm'

Most Norwegian compounds consists of noun-stems, as in example 6a to 6d. Some nouns have compound-only suppletive stems, as in example 7b.

Several of the examples to come, and all of the following percentages, are from a paper describing a compound-analysis algorithm used in a morphosyntactic tagger for Norwegian, based on the constraint grammar framework (Johannesen and Hauglin, 1998).

In Norwegian, compounds are generally written as a single word, with no spaces, as is the case for all the compounds in example 6 on the page before. In 75% of all cases, the compound consists of two juxtaposed stems, as in example 7, though the first stem of the compound can either be a normal stem as in example 7a or a suppletive stem that only occurs in compounds as in example 7b.

(7) Direct juxtaposition

a. Full stem

realfag
real+fag
'the natural sciences (with mathematics)'

b. Suppletive stem
Normal stem: *klær* (clothes, garment)
Suppletive stem: *kles-*
Examples:

i. *klesbransje*♢
kles+bransje
'garment industry'

ii. *kleshenger*
klær+henger
'coat hanger'

iii. *klesklype*
klær+klype
'clothes pin'

Sharp eyes might have noticed that *klesbransje* is marked as not occurring in the ENE dictionary. While the simplest explanation for this is that no dictionary can contain all words, and a slightly more interesting explanation is that the word in question was simply overlooked, it does make one wonder whether compounding with suppletive stems or just this particular suppletive stem is productive in Norwegian.

There are 17 words/compounds starting with *kles-* in the ENE dictionary (number of translations in parenthesis), *klesappretur*, *klesbanker*, *klesbevisst*, *klesbylt*, *klesbørste*, *klesdrakt* (6), *klesforretning* (2), *kleshenger* (3), *klesklype* (2), *klesmøll*, *klesplagg* (2), *klespoker*, *klesskap* (4), *klessnor*, *klesstativ* (3), *klestørk*, and *klesvask* (4), but none that start with *klær*[12]. Furthermore it is not difficult to generate transparent compounds beginning with *kles-*, like *klespoliti*♢ 'clothes police', *klesmafia*♢ 'clothes mafia' or *klesfiber*♢ 'fiber of clothing' that one can assume would be understandable.

Other suppletive stems are more of a variant spelling, for instance *bilde* 'picture' vs. *billed-*, where the ENE dictionary only lists compounds starting with *billed-* and cross-references *bilde-* to *billed-* instead of also listing compounds starting with *bilde-*.

With *hode* vs. *hoved-* one might ask if one is dealing with suppletive stems at all as the meaning of *hoved-* in the compounds generally is 'main' and not 'head'.

[12] *klær* is an interesting word in its own right, being plurale tantum. The idea of a single piece of clothing is expressed by the words *plagg* and *klesplagg*.

Compounds containing abbreviations are written with a hyphen as in example 8, and newer compounds often are as well.

(8) Hyphen

ABC-våpen
ABC+våpen

'ABC weapon'

A spelling-rule of Norwegian dictates that three or more identical adjacent letters are written as two, as shown in example 9 below. Furthermore, many Norwegian words can be spelled in more than one way, for instance the compound *andregradsligning* 'second-degree equation' can also be spelled *andregradslikning*, *annengradsligning* and *annengradslikning*.

(9) Spelling changes

busstasjon
buss+stasjon

'bus station'

The remaining compounds separate the two stems with an epenthetic sound, either *-s-* or *-e-*. Still staying with the analysis from Johannesen and Hauglin (1998), S-epenthesis is the most frequnt of the two, with 17%, while e-epenthesis account for the remaining 8%.

(10) Epenthesis

a. S-epenthesis

gårdshund
gård+hund

'farm dog'

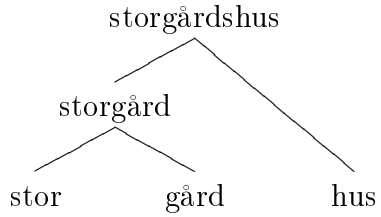
b. E-epenthesis

sauebonde
sau+bonde

'sheep farmer'

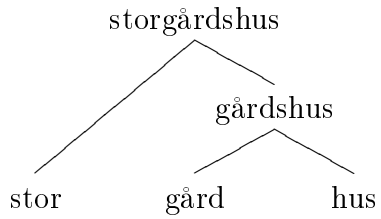
Finally, when more than two stems are involved, the compound can technically be split more than one way. Example 6e on page 42, *storgårdshus*♢, can be split as

(11) a. *storgårdshus*♢



'house of a large farm'

b. * *storgårdshus*♢



'large farmhouse'

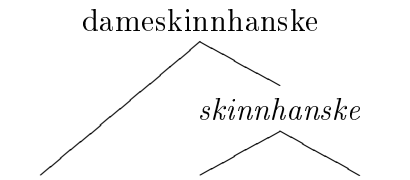
c. stort gårdshus
large farmhouse
'large farmhouse'

The split in example 11b is not reasonable because the epenthetic *-s-* tends to force a split when ambiguity arises. Another thing to note with these particular examples: the word *gård* is generally always followed by the epenthetic *-s-* if being a non-final stem of a compound, while *stor* never is.

An example like 6d on page 42 splits to ((gård+hus)+(tak+stein))+problem since both *gårdshus*♢ and *takstein* functions as stems on their own.

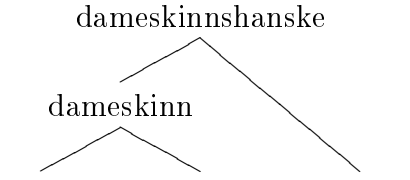
The next example is of a slightly more serious ambiguity, again showing the importance of the epenthetic *-s-*. While example 12a *can* be split both ways, the split as shown gives the most useful meaning for discourse. In contrast, example 12b would hopefully turn out to be less frequent in Norwegian texts. Another item in example 12a's favor is that its rightmost constituent has an entry in the E-N-E dictionary.

(12) a. *dameskinnhanske*◊



'women's leather gloves'

b. * *dameskinnshanske*◊



'gloves made of women's skin'

3.2.1.1 Guidelines for avoiding ambiguity in analysis

S-epenthesized compounds can be confused with non-epenthesized compounds when the first stem of the set ends in *-s-*, as in example 13.

(13) *klesvask*

a. kles + vask
N + N
'clothes wash'

b. ? kle + -s + vask
V + Ø + N
'clothe wash'

Potential ambiguity of compounds with e-epenthesis is due to most verbs in the infinitive ends with *-e-*, thus leading to situations like in example 14.

(14) *hoppetau*

a. ? hoppe + tau
N + N
'mare rope'

b. hoppe + tau
V + N
'jumping rope'

c. hopp + e + tau
N + N
'jump rope'

In addition there's the potential ambiguity caused by inflectional and derivational affixes (whether epenthesis is used or not) as seen in example 15. Examples like these can often be handled by preferring the analysis with the fewest stems, eliminating example 15c[13].

(15) Other ambiguity

a. slottsvinduene

slott -s vindu -ene
castle window -plural.definite
'the castle windows'

b. ? slottsvinduene

slottsvin due -ne
castle-wine pigeon -plural.definite
'the pigeons of the castle wine'

c. ?? slottsvinduene

slott svin due -ne
castle pig pigeon -plural.definite
'the castle-pig pigeons'

3.2.2 Equivalent constructions in English

Not all Norwegian compounds translate to compounds in English[14], as examples 18 to 21 on the next page demonstrate. Therefore it is more fruitful to

[13]Example 15b is marked as unusual/questionable by the author solely on an intuitive basis, although it can be mimicked by a rule that prefers compounds having stems of approximately the same length.

[14]Details of compounding in English are to be found in Downing (1977); Ryder (1994) and Huddleston and Pullum (2002, chapter 19, section 4).

look at what Norwegian compounds translate to in English[15]. The following examples are of Norwegian noun-noun compounds and their equivalents in English:

(16) noun + noun

a. *gårdshus*◇ - farmhouse

b. *bygård* - apartment building

(17) adjective + noun

a. *statsvitenskap* - political science

b. *forretningsstrøk* - commercial area

(18) noun + *of*-phrase

a. *produksjonsmidler* - means of production

b. *streikevarsel* - notice of a/the strike

c. *allemannsrett* - public right of access

(19) possessor noun + possessed noun

a. *vinterdag* - winter's day

b. *kvinnebevegelse* - women's movement

(20) single words

a. *arbeidsgiver* - employer

b. *hovedstad* - capitol

(21) other

a. *arbeidsro* - opportunity to work undisturbed

b. *skoleplikt* - compulsory school attendance

c. *skolevei* - way to and from school

d. *streikerett*[16] - right to strike

e. *kjøpelyst* - desire to buy

The examples 18c), 21a) and 21e) were found in the bilingual dictionary but not in Bokmålsordboka.

[15]Most of the examples were found in Hasselgård et al. (1998).

[16]In both this example and the next the first stem can be analyzed in two different ways, as **noun**+**-e-**+**noun** or **infinitive verb**+**noun**.

3.3 Machine translation of compounds

What follows is an overview of relevant previous work about machine translation of compounds using both the web as corpus and deliberately collected corpora.

3.3.1 Translation using a corpus and templates

Rackow et al. (1992) translated compounds from German to English in 1992. The paper includes rules for segmenting German compounds (section 2). As with Norwegian, in German the words of the compound are occasionally separated by epenthetic sounds, compare with example 14 on page 46:

(22) a. Geschichtsbuch
Geschichte+buch
'history book'

b. Geschichtenbuch
Geschichte+n+buch
'story book'

The translation itself is done by a compositional transfer module (section 3). A transfer lexicon from German compound structures to English equivalents was set up by analyzing a bilingual German-English dictionary (MRD Collins English-German). Furthermore, when a feature of the compound-stems differed between the source and target compound, the feature of the target stem won. Take for example the German compound *Industrieinformationen*, whose final stem is marked for plural. Since the English equivalent *industry information* is singular, a rule was added to the entry for *-informationen* ← *information* that the latter should always be singular.

By analyzing the compounds of the MRD they found that noun-noun compounds are the most frequent in both languages, with 54.4% German noun-noun compounds translating directly into English noun-noun compounds. Going from German noun-noun compounds to English adjective-noun compounds was correct in 17.2% of the cases, while translating to the English noun-*of*-noun constructions covered another 14.3% of the cases.

Due to more than half of the translations being noun-noun, this was made the default. Later in the project an experimental system was added to cover the remaining cases: other possible translations were checked in a corpus of 40 million words from *Washington Post* articles and the most frequent there was chosen.

When the *Washington Post*-corpus lacked the equivalent construction or had only one example, another corpus was used, a subset of the HANSARDS bilingual corpus.

If the first word of the source compound translated to both an adjective and a noun, they checked the HANSARDS to see whether the adjective- or the noun-form was most frequent in other compounds: *Umweltbewegung* becomes 'ecological movement', adjective+noun, instead of 'ecology movement', noun+noun, because `ecological` was more frequent than `ecology` in other compounds.

3.3.2 Using the web as corpus

Grefenstette (1999) used translation of compounds as an example to show the utility of using the world wide web as corpus. He split German compounds into two stems and also looked at Spanish two-word multi-word units. Each stem/word was translated to English, as shown for the German word *Appartementhaus* in example 23a and (b), then combined with each other as shown in example 23c. Finally he searched for each combination in the web search engine AltaVista and ranked the combinations by the frequencies AltaVista reported per combination.

(23) appartement+haus

a. Appartement → apartment, flat

b. Haus → chop, cut, house, rampage

c. All combinations:

i. apartment chop, apartment cut, apartment house, apartment rampage

ii. flat chop, flat cut, flat house, flat rampage

In order to evaluate, he only searched for transparent compounds that had known translations, using a dictionary as a gold standard. If AltaVista's top pick was the same as the known translation, that was counted as one correct translation. It is therefore a relevant question whether the compounds that are selected for a dictionary will pattern in the same way as previously unseen and unknown compounds, transparent or not.

For German to English the system was correct 87% of the time. However as he himself notes, there was only a single template used for English, namely noun followed by space followed by noun. Thus the experiment could not come up with constructions like those shown in the examples of subsection 3.2.2 starting on 47.

3.3.3 The feasibility of shallow processing and improving the selection algorithm

While Tanaka and Baldwin (2003a) is a study into the feasibility of shallow processing to translate compounds, Tanaka and Baldwin (2003b) primarily looks at selecting the best translation candidate and how to better evaluate them.

Japanese was chosen for two reasons: noun-noun compounds are frequent in Japanese[17] and English and Japanese are not related. The results will therefore reflect the qualities of the method instead of the similarities between the languages as would be the case if the language pairs were a Germanic language and English as in Rackow et al. (1992).

The first paper discusses several methods for translating noun-noun compounds. *Memory-based machine-translation* (*MBMT*) is one such method, utilizing a pre-computed lexicon of compounds. The most obvious way to populate such a lexicon is to look up compounds in a dictionary (which they call *dictionary-driven memory-based machine-translation* or $\mathrm{MBMT}_{\mathrm{DICT}}$).

Another is to extract compound pairs from a word-aligned bilingual corpus, a method they call *alignment-driven memory-based machine-translation*

[17]Tanaka and Baldwin claims an estimate of 10% of tokens are compound, and that slightly more than half of the compounds occur only once in the corpus.

($\text{MBMT}_{\text{ALIGN}}$). While word-aligned corpora a hard to come by, *comparable corpora* are simply corpora from the same domain, and Tanaka and Baldwin use a method from Cao and Li (2002) to derive translation pairs:

First, noun-noun compounds are extracted from a source-language corpus. Then, as in Grefenstette (1999), each noun stem in the compound is translated to English via a bilingual dictionary, and each left-hand side translation is combined with each right-hand side translation. Finally each pair is slotted into each translation template, and they use a corpus of the target language to choose the best translation. Tanaka and Baldwin calls this method *word-to-word compositional memory-based machine translation* ($\text{MBMT}_{\text{COMP}}$). The method is mainly limited by the number of translation-pairs in the bilingual dictionary.

The memory-based approaches are compared with *word-to-word compositional dynamic machine translation* (DMT_{COMP}) and *interpretation-driven dynamic machine translation* ($\text{DMT}_{\text{INTERP}}$). DMT_{COMP} only differs from the MBMT-variant by that the source compounds are not extracted from a fixed corpus but inputted directly. $\text{DMT}_{\text{INTERP}}$ is not a shallow method but analyzes the semantics and pragmatics of the compound to generate an interlingual semantic representation from which a target-language construction can be generated.

The rest of the paper looks at using only $\text{MBMT}_{\text{DICT}}$, only DMT_{COMP}, and the two cascaded, by first trying $\text{MBMT}_{\text{DICT}}$ and then letting DMT_{COMP} translate the remainder. The best results were achieved with using the two in combination, which consistently achieved more than 95% coverage though with lower accuracy than using $\text{MBMT}_{\text{DICT}}$ alone.

The papers does not mention the usability of non-classic corpora like the world wide web. The particular target corpus that was used was the *British National Corpus* (BNC), which was selected for its size and domain-inspecificity.

As for the evaluation of the system, instead of doing a manual evaluation by having a bilingual annotator rate the output for "plausibility" or "usability" they generated a gold-standard translation for each Japanese compound and compared the MT-translated compound to this gold standard. Depending

on the amount to translate this might be labor intensive, but the resulting verified translations can be reused, for instance to improve the coverage of a dictionary.

Tanaka and Baldwin (2003b) concentrate on the scoring algorithm for the ranking and selection of the best translations, calculating each probability p by a maximum likelihood estimate based on relative corpus occurrence.

The *corpus-based translation quality* (*ctq*) is given by equation 3.1.

$$ctq(w_1^E, w_2^E, t) = \alpha p(w_1^E, w_2^E, t) + \beta p(w_1^E, t)p(w_2^E, t) + \gamma p(w_1^E)p(w_2^E)p(t) \quad (3.1)$$

In this equation t is the translation template, mapping the structure of the source compound to the target structure. Next, w_1^E and w_2^E are the translation pairs. α, β and γ are fudge-factors for which the following holds: $0 < \alpha, \beta, \gamma < 1$ and $\alpha + \beta + \gamma = 1$. Each probability p is calculated according to a maximum likelihood estimate based on relative corpus occurrence.

Informally speaking, for a specific pair of stems originating from the source compound and a possible template: find 1) the probability of the entire translated structure in the target corpus, 2) the probability of the translated stem 1 and a random target-stem 2 using the same template and corpus, times the probability of a random stem 1 and the given translated stem 2 using the known template and corpus and finally, 3) the probability of translated stem 1 anywhere in the corpus times the same for stem 2, times the probability for the template regardless of stems.

3.4 Transparent non-compound MWEs

The prototype has been used to experiment with snowclones[18], a variety of cliched semi-fixed expressions with specific replaceable segments. These segments can be replaced by words that look almost the same written, sound almost the same spoken, or belong to the same part of speech, and more

[18]A term coined by Glen Whitman at 22:56:57 o'clock on Thursday, January 15, 2004, in Northridge, California. (Pullum, 2004)

rarely whole phrases. One example is "All your **base** are belong to **us**.", meaning approximately 'we control/possess all your (military) bases'. This catchphrase still has over 500000 hits on Google. Variations include "All your base are belong to **U.S**", "All your **bias** are belong to us" and "All your **office** are belong to **me**".

Snowclones
men are from Mars, **women** are from Venus.
guns don't **kill** people. People **kill** people.
Elvis has left the building.
pink is the new **black**.
play it again, **Sam**!
Houston, we have a problem.
All your **base** are belong to **us**.
Frankly, my dear, I don't give a **damn**.
Gag me with a **spoon**.
Have **gun**, will travel.
Holy **X**, Batman!
I'm not a **doctor**, but I play one on TV.
I, for one, welcome our new **insect** overlords.
Eskimos have **three hundred** words for **snow**.
In space, no one can hear you **scream**.
It's people! **Soylent Green** is made of people!
Let **bygones** be **bygones**.
Show me the **money**!
The **devil** made me do it!
What is this "**sixties sitcoms**" of which you speak?
When **guns** are **outlawed**, only **outlaws** will have **guns**.
Will the real **X** please stand up?.
Enige og tro til **Dovre** faller.

Table 3.1: A list of some snowclones spotted since the beginning of 2004. The bolded text are the replaceable segments, the particular words used are believed to be the original ones. While most of them is based on fixed phrases, the two with **X** originate from TV-series where the **X** varied with each episode. The very last example is a genuine Norwegian snowclone discovered by the author.

The study of snowclones is so recent that neither Moon (1998) nor Sag, Baldwin, Bond, Copestake, and Flickinger (2002) mentions them, but it is

clear that they would sort under what Moon calls *pragmatically marked formulae*. While some are so old as to be invisible (e.g. "**Stupid** is as **stupid** does."), others still take much of their connotations from the original source, like the afore-mentioned "All your base...".

As can be seen from the table 3.1 on the facing page, many of the snowclones ultimately derive from popular TV-shows and movies, as for instance "I, for one, welcome our new **insect** overlords." hails from the cartoon TV-series *The Simpsons* while "It's people! **Soylent Green** is made of people!" comes from the motion picture *Soylent Green*.

Example 24 shows the queries to use to search for a particular snowclone. The first two works best on Google, the last should work on any search engine that has phrase search and negation.

(24) "Enige og tro til **Dovre** faller."

a. `"enige og tro til * faller" -dovre`

b. `enige-og-tro-til faller -dovre`

c. `"enige og tro til" faller -dovre`

If the target language has a construction with a similar meaning and the same number and types of replaceable parts, these can be translated in the same vein as transparent compounds. However, while there are at least one specimen of a Norwegian snowclone, as shown in example 24, it might be that snowclones are mostly a phenomenon of English-speaking and English-influenced popular culture, or narrow genres like opinion-pieces and editorials. There are few if any snowclones in the texts in the domain of the LOGON project, so further experiments have been put on hold.

Chapter 4

> . . . the Net is a world of ends. You're at one end, and everybody and everything else are at the other ends.
>
> Searls and Weinberger, 2003

Search engines and the translation of compounds

The internet is a world of ends: the computational linguist is at one end, studying linguistic data in form of texts or recordings that is located at one or more of the other ends. This underlines the fundamental difference between a deliberately collected static corpus (e.g. the WSJ corpus) and the texts to be found on the internet: the computational linguist has no control over the source of the texts at the other ends; who added them, what they contain or how long they will remain. This is, simultaneously, both the the biggest strength and the biggest problem of using the internet as corpus: it is language as it is used at the moment, completely unfiltered and unprepared.

Fortunately, there are tools to zoom in on interesting textual phenomena in texts stored on the internet, namely web search engines. Unfortunately, web search engines are not designed for use by computational linguists but for use by anyone and anything capable of processing text.

This makes using a web search engine quite different from using a static corpus. For one, even though a web search engine (like Google or Yahoo! Search) caches some of the documents it stores in its index, a search engine does not work on the same principle as a database. For instance, search engines increasingly try to guess what the searcher wants, attempting to find relevant documents with queries containing only a single search term. While

a database aims to retrieve all records defined by a query, (say, all documents containing the token NLP), a search engine will attempt to retrieve links to all documents *relevant* to a query but more importantly it will attempt to *rank* them according to a *relevancy* algorithm or measure. The exact working of the latter is both subject to change over time and a closely held secret of the search engine providers.

Therefore, we will first need to dive into some of the theoretic[1] aspects of search engines and web search. With the basics in place I will then demonstrate a practical implementation of an automated system for translating compounds using search engines instead of a corpus, `ReCompounder`.

4.1 Using search engines for linguistics

The two leading search engines, both in terms of users and index-size, are Google at `http://www.google.com/` and Yahoo!'s search engine Yahoo! Search at `http://search.yahoo.com/`. Both AltaVista, which have been used before for a similar purpose (Grefenstette, 1999; Keller and Lapata, 2003), and Alltheweb was purchased by Yahoo! in 2004 and are now front-ends to Yahoo! Search.

Not only are the indexes of today much larger[2] than they were in the early days of the internet[3], but the query syntax and ranking algorithms are also different from 1999. In addition, the search engines can now be accessed and used through stable APIs (Application Program Interface). Analysing a search engine webpage directly is therefore no longer necessary.

4.1.1 Definitions

Before we continue it is necessary to agree on the definitions of several terms. The following definitions will be used throughout the remainder of this chap-

[1] or as theoretic as something can get when a laboratory is only a web browser away.

[2] As of September 27, 2005, index-size is no longer given on the search-pages (Sullivan, 2005).

[3] Yahoo! Search's index was given as containing 19.2 *billion* web documents in August 2005 (Mayer, 2005)

ter and are therefore given up front.

document Anything that has an url, be it a webpage containing text, a picture, an audio recording, ia piece of software, compressed files, encoded files, raw binary data...

frequency (of a query) The number of documents in the index that can be said to be relevant to the query in some way. For instance, relevant by containing one or more tokens of the query in the content, url or metadata, linking to or be linked from another document that is relevant to the query, or some other definition of relevancy that is specific to the particular search engine. As of this writing, if the frequency is higher than 1000 it is estimated, otherwise it is accurate. The frequency is usually shown in the upper right quadrant of the first screen of results, just before the hits. The Google API also marks whether the returned frequency is estimated or not.

ghits *Google hits*: The estimated frequency of a query in Google, given before the results themselves.

hit A link listed in the results. It might be just an url, or in addition it might contain a title and lines of context for the query, a link to a cached version, date of last retrieval, size of document and other hit-relevant or search engine specific links.

hit context A few lines of text from the hit if the content of the hit was somehow relevant to the query, or the first lines of the hit if some other relevancy measure was used.

index (noun) The sum total of documents a search engine knows about. There are two types of documents in an index: *indexed documents*, where the contents of the document is known, and *known documents*, where only the url is known. When used in NLP it's the indexed documents that are of interest.

index (verb) To add a document to the index, either by adding its content and url (for indexed documents) or just its url (for known documents).

link See url.

page scraper A program that downloads a given page and then tries to parse it to extract wanted information. Synonyms: *web scraping*, *site scraping*, *screen scraping*

query A text-string of one or more tokens (words) and operators (e.g. conjunction, disjunction, negation, limiting to search in titles or on specific sites).

ranking Algorithm that orders the hits in the results according to the search engine specific relevancy measures. The first hit of the results is also the highest ranked.

results Shorthand for *page of results* or *batch of results*. That is, a web page given by a search engine in response to a query, or data fetched through a search engine API in response to a query. If the query has a high frequency (many ghits or yhits) the results will generally contain 10 hits[4]. It is possible to ask for the next 10 hits of a query if such exist, this leads to the retrieval of another batch of results.

search (noun) One round of querying a search engine and retrieving the results. Asking for the next 10 results means another search.

url An address to something reachable on the internet, though in the case of web search engines usually limited to documents served over the *http*-protocol.

yhits *Yahoo! Search hits*: The estimated frequency of a query in Yahoo! Search, given before the results themselves.

4.1.2 What is possible

All the definitions above will be demonstrated in the following section, which describes a technique that is very useful for linguistics, namely phrase search. The same section will also discuss phenomena common to all types of search.

[4]Some search engines (e.g. Yahoo! Search) allow increasing this number.

4.1.2.1 Phrase search

The syntax for a *phrase search* is one or more words enclosed in double hyphens: `"phrase search"`. One might think that this searches for exactly the words in the exact same order as in the query, but this is not quite the case[5].

The ten first hits when searching for the hypothesized compound "repair truck" on Google is given in listing 4.1 below:

```
1  Web  Results 1 - 10 of about 86,800 for "repair truck". (0.33 seconds)
2
   Shows A - Z : Car Care & Repair : Truck Utility Upgrades : DIY Network
4  Projects: Installing a Ladder Rack/Installing a Truck-Bed Tool Box/Installing a
   Roll-Top Bed Cover & Tailgate Lock/Installing a Bed Extender/Adjustable ...
6  www.diynetwork.com/diy/shows_ccr/episode/0,2046,DIY_14271_25006,00.html - 44k -
   Cached - Similar pages
8
   Rhinelander auto repair, truck accessories, D and J Auto, Truck ...
10 Providing Rhinelander, Minocqua, Pelican, Pine Lake, Newbold, Three Lakes,
   Crescent, Woodruff, Sugar Camp, Nokomis, Hazelhurst, Lake Tomahawk, ...
12 www.explorewisconsin.com/DJTruckandEquipmentRepair/ - 21k - Cached -
   Similar pages
14
     Jackson body shops, Milwaukee auto body repair, truck repair ...
16   John's A1 Auto Body provides auto body repair, truck repair, auto restoration,
     stainless steel exhaust systems, auto paint, auto accessories, car repair, ...
18   www.explorewisconsin.com/JohnsA-1AutoBody/ - 21k - Cached - Similar pages
     [ More results from www.explorewisconsin.com ]
20
   REME Truck 3 ton 4 x 4 GS MT Repair Bedford RL
22 ... chassis but the Bedford chassis was selected solely for the MT repair truck.
   ... new NATO standards and it became Shop Automotive Repair Truck Mounted. ...
24 www.rememuseum.org.uk/vehicles/mwv/vehbedmt.htm - 6k - Cached - Similar pages

26   REME Truck 3 ton 4 x 4 GS Commer Q4
     The Museum has an example of the Telecommunications Repair truck which is ...
28   The other museum example is the Fuel Injection Equipment Repair truck. ...
     www.rememuseum.org.uk/vehicles/mwv/vehcomq4.htm - 8k - Cached - Similar pages
30
   Truck Parts & Service in Lafayette, LA on Lafayette Citysearch ...
32 RJ's Auto & Truck Repairs Incorporated; Auto Repair, Truck Parts & Service.
   1.25 miles ... Mike's Car Care Center; Auto Repair, Truck Parts & Service ...
34 lafayette.citysearch.com/yellowpages/directory/Lafayette_LA/10/635/page1.html
   - 106k - Cached - Similar pages
36
   Automotive, Service & Repair, Trucks, Brooklyn, New York, NY ...
```

[5]For more examples see Keller and Lapata (2003).

```
38 Categories: Auto Service & Repair, Trucks Service & Repair, Truck Refrigeration
39 Equipment Service & Repair, Truck Refrigeration Equipment Retail ...
40 www.superpages.com/yellowpages/
   CP-Automotive%5EService+&+Repair%5ETrucks/S-NY/T-Brooklyn/
42 - 148k - Cached - Similar pages

44   Automotive, Service & Repair, Trucks, Honolulu, Hawaii, HI ...
     Categories: Trucks Service & Repair, Truck Transmissions Retail, Marine
46   Engines & Transmissions Retail, Diesel Engine Parts Sales & Service ...
     www.superpages.com/yellowpages/
48   CP-Automotive%5EService+&+Repair%5ETrucks/S-HI/T-Honolulu/ - 178k -
     Cached - Similar pages
50   [ More results from www.superpages.com ]

52 Course Descriptions T - Sault College
   They will review the principles of electrical circuit schematics and use them to
54 diagnose and repair truck and coach electrical systems. ...
   www.saultc.on.ca/depts/Calendar/Coursedest.htm - 34k - Cached - Similar pages
56
   EngNet - Hydraulic Cylinder Design, Manufacture and Repair ...
58 Fully equiped facilities to repair truck mounted hydraulic cranes Hydraulic
   Lifting Equipment Repairs and Maintenance Fully equiped facilities to repair ...
60 www.engnetglobal.com/c/c.aspx/DEN010/products - 10k - Cached - Similar pages
```

Listing 4.1: "repair truck" in Google with line numbers and highlighting added for convenience.

Line 1 has the ghits, as defined in section 4.1.1 on page 58.

Only on lines 22, 23 and 27, 28 do we find genuine compounds. Elsewhere, "repair" is either a verb like in lines 54 and 58, or is separated from "truck" by punctuation like in line 45. In fact, by quickly browsing through the results-pages it becomes evident that the large majority of hits is for **repair, truck** and therefore not a compound. Furthermore, it turns out that Google only lets you see 630 total hits for this particular query even though a much higher number is estimated on line 1.

The results of the same example on Yahoo! Search is given in listing 4.2:

```
1 Results 1 - 10 of about 153,000 for "repair truck" - 0.10 sec. (About this page)
2
  WEB RESULTS
4
    1. RELIABLE DIESEL TRUCK & TRAILER REPAIR Newburgh New York 12550 [oo]
6     LOCATED IN NEWBURGH NEW YORK. WE OFFER 24HR ROADSERVICE, ON SITE FLEET
      REPAIRS, REPROGRAMING, DIESEL REPAIR, AND SERVICE FOR TRACTOR TRAILER TRUCK.
8     www.truckroadservice.com - 37k - Cached - More from this site - Save - Block
    2. Martin's Truck Repairof Windsor, California [oo]
```

```
10      Martin's Truck Repair specializes in truck repair services and offer
11      professional and quality truck repair and parts service everytime.
12      www.martinstruck.com - 11k - Cached - More from this site - Save - Block
    3. Truck & Trailer Repair [oo]
14      TDC Road & Truck service, based in Dayton Ohio , offers 24-Hour road service
        and heavy Duty Towing. We also have a full service shop facility.
16      tdcroadandtruckservice.com/repair.html - 10k - Cached - More from this site
        - Save - Block
18  4. Auto Repair and Maintenance articles at 10W40.com [oo]
        Auto Repair and Maintenance articles
20      www.10w40.com/home/auto_repair.asp - 22k - Cached - More from this site
        - Save - Block
22  5. Martinsville, OH Truck Refrigeration Equipment Service & Repair [oo]
        Find the Best Truck Refrigeration Equipment Service & Repair in
24      Martinsville, Ohio at SuperPages.com. SuperPages from Verizon has listings
        for many more Martinsville businesses.
26      yellowpages.superpages.com/listings.jsp?...&R=N&search=Find It&SCS=1 -
        More from this site - Save - Block
28  6. Mike's Truck & Auto Service [oo]
        Truck and automotive services and repairs. Full maintenance services. Gas
30      and diesel engine repair and diagnostics. Same day service on most repairs.
        ASE certfied.
32      www.mikestruckandauto.com - More from this site - Save - Block
    7. part repair truck [oo]
34      site list on part repair truck . ... Useful Info On part repair truck :
        Monday 10th of October 2005 04:57:09 AM ...
36      truck-resource.acdom.com/part-repair-truck.html - 16k - Cached -
        More from this site - Save - Block
38  8. panel repair truck [oo]
        site list on panel repair truck . ... Useful Info On panel repair truck :
40      Monday 10th of October 2005 08:14:45 PM ...
        truck-resource.acdom.com/panel-repair-truck.html - 16k - Cached -
42      More from this site - Save - Block
    9. auto repair truck [oo]
44      All internet collected resources on auto repair truck . ... Useful Info On
        auto repair truck : Thursday 13th of October 2005 01:03:32 PM ...
46      truck.acdom.com/auto-repair-truck.html - 16k - Cached - More from this site
        - Save - Block
48  10. chevy repair truck [oo]
        All internet collected resources on chevy repair truck . ... Useful Info On
50      chevy repair truck : Friday 14th of October 2005 11:31:40 PM ...
        truck-info.acdom.com/chevy-repair-truck.html - 15k - Cached -
52      More from this site - Save - Block
```

Listing 4.2: `"repair truck"` in Yahoo! Search with line numbers and highlighting added for convenience.

As with Google, the yhits are to be found in line 1. Yahoo! Search shows a maximum of 1000 hits, even though as with Google, the estimated frequency

is much higher. Furthermore, only from line 33 do we actually find **repair truck**.

A possible reason for the first hit to be included is that the string "diesel repair, truck repair" is found in a section of the source HTML that is not rendered on the visible page, thus visible to the search engine but not to a human user without close examination of the source. The second hit is more interesting: it does not contain **repair** directly followed by **truck** anywhere, not even separated by punctuation.

Another important thing to keep in mind is that the various search engines do not necessarily index an entire document. It is known for instance that Yahoo! Search indexes less of a PDF-file than Google (Véronis, 2005b). This means that tokens that are to be found in a document after the cutoff will not be searchable. This is especially relevant when searching for scholarly articles as these often are stored as PDF-files when online, making it necessary to add a section of relevant keywords fairly early in the paper so that the paper can be found regardless of its length.

To sum up so far:

- One cannot access more than 1000 hits per query, regardless of the estimated frequency of the query.
- Both Google and Yahoo! Search completely ignore punctuation, making a phrase search less useful, as seen in both listings above.
- A page need not include the phrase at all, as illustrated by the second hit for **repair truck** in listing 4.2 on page 62.
- A page or document *known* to contain a term might not turn up as a hit at all if the term is after the indexing cutoff.

Because of this, it is necessary to check the hits. It is not possible to check every hit: only the first hundred results-pages for a maximum total of a thousand hits are made available to us. When querying a search page directly it is possible but somewhat slow to check all the provided pages of results. However, when using the APIs, each page of results decrements the

number of remaining searches by one, even if we are looking at lower ranked hits of the same query. Therefore it is not desirable to spend several searches on a single query when studying queries in batch, as we'd quickly run out of unused searches. Finally there's a limit on the time and resources that can be used to translate a single word in a machine-translation system.

Taking all of this into account, the prototype to be described later only checks the hits of the first page of results, for a total of ten hits, as will be discussed in section 4.2.5 on page 84.

4.1.2.2 Boolean operators

Both search engines combine the terms of the query with implicit Boolean AND. This differs from earlier search engines that used Boolean OR. However, as seen in the previous section, a hit need not contain the query in its context at all. Worse, as was discussed by Véronis and shown in the examples of the word-hyphen operator to come, the logic used in combining queries need not necessarily be Boolean at all but pseudo-Boolean. As mentioned there, this might be due to operator precedence differing from what the user expects or more weight being given to the first token in the query.

4.1.2.3 The undocumented word-hyphen operator in Google

Table 4.1 on the following page shows the results from experimenting with connecting query words with a hyphen in Google and Yahoo! Search. All queries were done within a few minutes of each other so as to improve the chances of getting the same frequencies, and the word to query for was chosen because of its low frequency, therefore preventing the search engines from estimating the frequency.

The first thing to notice is that the hyphen clearly doesn't mean the same for the two search engines (the first half of the table), the second that Yahoo! Search either lends extra weight to the first token of the query or that the operator precedence (which operator is acted on first) is different from Google. Otherwise the eleventh and twelfth rows would both have the same yhit as row one, and the last two rows would have the same yhit as

	Google	Yahoo! Search
`pagescraper`	243	103
`"page scraper"`	329	300
`page-scraper`[a]	361	1,500,000
`"page-scraper"`[b]	329	307
`page-scraper -pagescraper`	324	1,500,000
`pagescraper -page-scraper`	187	23
`"page scraper" -pagescraper`	328	295
`pagescraper -"page scraper"`	187	93
`"page scraper" OR pagescraper`[c]	361	356
`pagescraper OR "page scraper"`	361	354
`pagescraper OR "page scraper" -"page scraper"`	187	103
`"page scraper" OR pagescraper -"page scraper"`	187	356
`pagescraper OR "page scraper" -pagescraper`	324	356
`"page scraper" OR pagescraper -pagescraper`	324	301

[a] Judging from the ghits, Google seems to treat this query equivalent to `pagescraper OR "page scraper" OR "page-scraper" OR "page? scraper"`, where ? is any single punctuation-character. Very few of the yhits actually contain **page** directly followed by **scraper**, showing that a binding hyphen is used very differently in Y and G.

[b] The first ten results in Google only includes **page scraper** and **page-scraper** while Yahoo! Search's are identical to the yhits of **"page scraper"**.

[c] it would seem that Google's word-hyphen operator is defined as **"page scraper" OR** `pagescraper`.

Table 4.1: The frequency of `page scraper`

`"page scraper"` in the second row.

In Google, the pattern `word1-word2` seems to be doing much the same as the pattern `word1word2 OR "word1 word2"` except for getting more hits. However `"word1-word2"` gets much fewer hits than `word1-word2`.

In Yahoo! Search, the pattern `word1-word2` gives much more yhits than `"word1-word2"`. The first pattern seems identical to the pattern `word1 word2` while the second seems identical to `"word1 word2"`.

Common for both search engines is that the results returned need not contain the dash, and might be separated by other punctuation instead. As mentioned earlier, this also seems to be the case whenever there are more than one word to search for.

To conclude, it seems that using web search to discover whether a compound is written with or without a hyphen is futile at this time.

4.1.2.4 Other common syntax

In addition to the above there are several colon-operators e.g. `site:` and `link:`. Both of these are useful if one wants to attempt to limit a search to a particular domain in the linguistic sense. By using `site:host` the search is limited to a particular host (domain in the Internet sense). By using `link:` the search is limited to pages that link to the given url. There are examples of both below[6].

(25) a. Ignoring a site:
`compound -site:wikipedia.org`

b. Limiting to a site:
`compound site:wikipedia.org`

c. Ignoring pages containing specific url:
`compound -link:wikipedia.org`

d. Counting pages containing specific url:
`compound link:wikipedia.org`

[6]Section 4.1.4.1 on page 73 discusses when and why the `site:` and `link:` operators should be used and gives a list of hosts to filter out.

e. Combining both operators:
`compound -site:wikipedia.org -link:wikipedia.org`

Web host-names are used by roughly three different groups: individuals, which include family-domains and so-called vanity domains; organizations, which include most business-sites; and groups of organizations, which include for instance universities, blog hosts, web message boards, large corporations and Internet service providers (ISPs). Hosts from the the two first groups can usually be assumed to be domain-limited since these are written by only a few individual people each. The last group usually have several internal domain-limited sub-sites, therefore, greater care must be taken when using `site:` or `link:`.

There are also other colon-operators but these did not prove useful for the purposes of this thesis and will not be discussed.

4.1.2.5 "Magic" queries

Giving Google a query like `seven feet in meters` will currently be interpreted by the Google calculator in addition to the regular search engine. This leads to a special zeroth hit in the web interface that is listed before the ten ordinary hits. For the query in question the zeroth hit is **seven feet = 2.1336 meters**. The calculator can also be used for simple arithmetic, for instance will `seventeen times three` give a zeroth hit of **seventeen times three = fifty-one**.

The Google calculator is the first instance of "magic" queries in the large search engines. It is easy to imagine future extensions, for instance lookup in an encyclopedia if a query, for instance `"Mornington Crescent"`, can be interpreted as a name of a person or place.

4.1.2.6 The impossible: inherent linguistic limitations

When using web search engines, we must never forget where the content comes from. The readers and writers of documents on the web are subsets of people that can afford to and have the time to access it: there is thus a systematic bias. Each hostname is potentially a domain unto itself, their

content having been collected or made by individuals with sometimes hidden agendas. Furthermore, a document might have been written or translated by someone with only a basic grasp of the language, leading to a high number of spelling errors and more unusual word choices than normal.

4.1.3 Ranking and relevance

Regardless of the query, it is clear that the results are ranked in some way, since searching for the same query at some later date will yield approximately the same results in approximately the same order. We must also assume that the ranking is an attempt to show the most relevant results earlier than the less relevant.

However, most of us do not work for the search engine companies and hence do not have access to any proprietary information about the search engines used. This means that we don't know the actual heuristics and algorithms that are used to rank the hits by relevancy. Neither do we know the actual number of total documents indexed, the number of documents containing information relevant to the query, how the query is interpreted, how relevance is computed, how the frequency given is estimated from the actual frequency and how the data is presented to the user. Furthermore, all of the preceding are subject to change without notice.

We can take the empirical approach: input several queries, study what is returned in each case, and then try to find a hypothesis that fits the data. We can try to use old, well-tested relevance measures like precision and recall, or we can resort to educated guesses.

Ignoring the last possibility completely, next we'll see how or if precision and recall fit with the realm of web search, then we'll go back to discussing relevance and ranking.

4.1.3.1 Precision and recall

Web search excels at recall. If there exists a document in the index that contains one of the words of the search query, or a link to a document that is not indexed but was found to contain one of the words of the search query,

or it fulfills other criteria only known to the designers of the search engine in question, a link to the document will be returned. These criteria doesn't necessarily lead to the same results as what a language researcher would call a relevant hit. The problem is how to improve precision: how to determine which hits are most relevant (most precise) and how to define and filter out unwanted hits.

We'll use the textbook definitions of precision and recall, more specifically the textbook by Jurafsky and Martin (2000, p. 652):

P : precision

R : recall

n : number of documents returned

r : number of relevant documents returned

t : total number of relevant documents in the collection

$$P = \frac{r}{n}$$
$$R = \frac{r}{t}$$

If there are fifty million documents relevant to a query in the index (t), and the search engine estimates the frequency to be forty million (n), we'd still not have enough information to calculate neither precision nor recall. Neither would this number be relevant to the user, as he or she would only be allowed to access the thousand highest ranked documents anyway.

As we sit at our end of the web, studying what the search engines tell us about the ends they know about and allow us to know about, we cannot know t, nor the total number of documents in the collection, since this number is no longer publicized. The only number we are given is the frequency (ghits or yhits), which is n, and of n, we can only check the first thousand. Ergo, using the standard formula for precision and recall becomes rather difficult.

We now go back to studying search engines empirically.

4.1.3.2 Ranking

Due to the large amounts of documents that are indexed by web search engines, and due to all ghits and yhits above 1000 being estimates, if we want to empirically discover how ranking and relevance works for a particular search engine we must choose queries with less than 1000 hits. Finding a single word that has less than a thousand hits can be difficult[7], so it might be better to use compounds or bigrams that is expected to be rare.

Search engines evolved out of the field of *information retrieval* (or *IR*), and the early Internet search engines used a mainstay of information retrieval, the *inverse index*[8]. While a right-way index uses a document-id as a key to retrieve a document, an inverse index uses the contents of a document as keys to retrieve the ids of documents that contain the keys.

As an example, if the word form "embalm" exists in documents A, B and C, while the word form "water" is found in documents B, C, D and E, looking up "embalm" will return the documents A, B, and C (or addresses to these documents) and not documents D or E. Searching for documents containing both "embalm" and "water" will then retrieve documents B and C.

However, a document only containing "embalm", or only "water", might still be relevant to the person making the search[9], so for a search engine it pays to *also* return documents A, D and E. Since in order to find out which documents contained both terms it was necessary to recall which documents contained one of the terms, the lists of documents have already been made, leading to recall becoming cheap.

This is where ranking comes in: B and C, containing both terms, will outrank A, D and E and hence be listed earlier in the results. However, if we do attempt to read the mind of the searcher, since "embalm" was listed *first* of the two terms, we assume that this is the most important term and thus rank documents containing "embalm" higher than documents that don't.

[7]Unless we're using words from a language with only a small web presence and thus few indexed documents.

[8]Whether search engines still use simple inverse indexes we cannot know since this information is proprietary.

[9]After all, mind reading is rather hard.

The final, partial ordering of returned documents might then be B and C, then A, then D and E.

For an actual search engine, looking for documents relevant to both "embalm" and "water" of our example will result in more than 100000 ghits and more than 60 000 yhits, meaning that the combination is insufficiently rare to be useful for studying how the ranking works. Another complicating factor is how ranking is adjusted by the relative popularity of the documents, we will not speculate about that here.

4.1.3.3 Relevance

What I consider to be the key insight to *why* the search engines' definition of relevance can be less useful to a linguist is that there are at least two different types of search:

1. *Exploratory search*, for instance googling yourself (using your own name as a phrase search to see what the search engines have stored about you), or searching for hotels at some resort where you'll have your vacation, or studying bias by typing in a single, homonymous word[10] or any other query of curiosity. Exploratory searches can answer a question.

2. *Recovery search*, where the searcher already knows what to expect, searching for a book or paper or some other piece of data that is known to exist using a query that is known to be part of the wanted documents. The query itself is the title of the document, or the author or creator, or a phrase that occurs in it. In addition to being capable of answering questions, recovery searches can confirm the answer to a question.

What is relevant for exploratory search is less relevant for recovery search. For exploratory search, high recall is an asset; for recovery search we want precision. Second-guessing the searcher improves exploratory search but using the web as corpus, finding frequencies, is in the realm of recovery search.

[10] As of 2006-03-21, 8 out of 10 hits for `apple` was for the computer company, not the fruit.

4.1.3.4 The quality of recovery search

The results of a web search presents a snapshot of the document index, query syntax and ranking algorithm at the moment of search. A later search might not return the same results, ranking or frequency, but the relative frequency of the query as compared to another query changes only when the index is updated, thus slowly. This means that differences between the frequencies of the results of many searches done in a short and limited period of time will be approximately the same, and that this frequency is trustworthy.

However, since web search have been aimed more towards exploratory search and the results as is might contain hits that are irrelevant to our recovery search, it becomes necessary to check the results somehow. In any case we can only check the first thousand. It is an unanswered question whether the cost in time of downloading and then checking the first thousand yields a better quality than just checking the provided context of the first thousand or a subset thereof.

The next section will look at prominent sources of noise that are due to people and not technology, and how to take this noise into account in order to improve relevance.

4.1.4 Web-specific sources of noise

In addition to the noise caused inherent to the search engines themselves, like ignoring punctuation in queries and including pages that does not contain the query at all, there are two phenomena that one is spared from in consciously and deliberately collected corpora: duplicates and spam.

4.1.4.1 Duplicates

Duplicates on the web are mainly due to content that can be freely copied are being hosted at several different sites. Well known duplicates include the entirety of Wikipedia, which aims to be a comprehensive and free encyclopedia; the entirety of the Websters dictionary from 1913 which is in the public domain; the entirety of Project Gutenberg which is a collection of digitized

books that are in the public domain and the Unix `ispell`-files which are lists of English words to be used for spelling correction. Duplicates are also caused by the same site being known under several different names.

The result of such duplicates are that words, expressions, bigrams, correspondences etc. that occur in the duplicates will have an artificially increased frequency compared to text that are not freely copyable.

A good illustration of the problem of duplicate content is exemplified by an NCSA[11] that aimed to compare the index-sizes of Google and Yahoo! Search as it was in the middle of 2005. The researchers, two students of Professor Vernon Burton, made a query out of two low-frequency words and compared the resulting ghits and yhits, concluding that Google had the largest index[12]. However, as Véronis (2005c,d,e) points out, this will bias the results heavily towards wordlists like the before mentioned `ispell`.

Even excluding a third low-frequency term from the search query will still bias the results towards wordlists and auto-generated spam-pages, as examining the results from the query `switchers trophoblast -agnus` will prove, which as of 2005-12-07 returns 8 wordlists or auto-generated pages out of the first ten possible when searching on Google. In the end, a new version of the paper was released, with the above fix included, and without NCSA or the name of Professor Burton mentioned anywhere and including the following disclaimer[13]:

> [...] A verification study is currently in progress that addresses the presence of "wordlists" and "dictionaries" in the search results that many rightly point out could count as a source of bias. The new study filters out any dictionary or wordlist results. Preliminary results (from 7000 test queries) indicates that the results of this verification study confirms the conclusions of this study, but

[11] It was first claimed to be an NCSA study, and later claimed that it wasn't, muddying the waters further.

[12] The following url links to the study as it is today: `http://vburton.ncsa.uiuc.edu/oldstudy.html` (accessed 2006-01-09) Note that this is not the original, which can currently only be found via Véronis (2005d).

[13] See Véronis (2005e) for the entire disclaimer.

final results are still forthcoming.

While this debacle didn't teach us much about how to discover which search engine has the largest index, it certainly serves to remind the prudent researcher to take duplicate pages into account when using search engines to acquire frequency information.

Known duplicate sites can be excluded by using the aforementioned `site:`-operator: `linguistics -site:wikipedia.org -site:answers.com -site:references.com` will look for mentions of linguistics and filtering out pages on wikipedia and two of its known duplicates at the same time. While most duplicates of wikipedia includes a link back to it, thus making it possible to also filter by adding `-link:wikipedia` to the previous query, other categories of duplicates frequently do not.

4.1.4.2 Spam

Spam on the web are pages and sites that are built solely to trick search engines into giving them a high ranking. When a user is led to select a spam-page instead of a page with legitimate content, the spammer earn money by having advertisements that pay per view or by users fooled by a scam hosted on the page in question.

This can be done by purchasing a site that has a good ranking for a particular query and then replace all the content with advertisements, or by creating *link-farm*s: pages and sites that are designed to rank high for a particular query that link to other pages and sites that are similarly designed for the same query.

Prior to Google's PageRank-algorithm, which ranks pages by how many other pages link to it, it was common to utilize *keyword poisoning* or *keyword spamming*: keywords that would lead to a high rank was hidden in the header of the HTML-file or given the same color as the background, and thus invisible to a user but not to a search engine.

Web-spam are of interest to computational linguists for at least three reasons:

1. Computational linguists are hired by the search engine companies to discover ways to find spam-pages so that these can be removed from the results.

2. Computational linguists are hired by spam-firms in order to design spam-pages that are not found by the computational linguists of point 1).

3. Computational linguists that are neither in category 1) nor 2) above, who try to use search engines for computational linguistics, have to take into account the added noise because of the spam.

Yet another way of poisoning the index of a search engine is by adding spam to pages that can have content added to by casual users, as for instance wikis like wikipedia, or any place a user can leave a comment, be it in blogs like the snippet below, in online newspapers or in guestbooks.

```
shear vexing,oscillations Quichua licensed Pernod Andrea. poker games
red hot poker tour poker games red hot poker tour
http://poker-games.standard-poker.com/
http://poker-games.standard-poker.com/ dislocations,Creon
detects?creation fiddling pacific poker pacific poker
http://pacific-poker.mynet-poker.com/
http://pacific-poker.mynet-poker.com/ isolate!cancellations heads up
poker strategies wsop heads up poker strategies wsop
http://wsop.ws-op.com/ http://wsop.ws-op.com/ cabins assassins
acceptably basketballs poker games poker games
```

Listing 4.3: Example of low-quality comment-spam found in a blog, link-poisoning for poker-sites.

Site-administrators can prevent spam as in listing 4.3 on user-contributed content by automatically adding the *rel="nofollow"* HTML-attribute to all link-tags in the user-contributed content as it is being stored, since it can be hard to discover what is user-contributed or not after the fact. Search engines honoring the tag will then not increase the ranking for the link in question and thereby render the spam impotent. Preventing anonymous users from adding content is only a stop-gap measure as the spammers figure out how to register accounts automatically.

While spam in email often try to sell a product directly, and is designed by people to sneak through email spam-filters, web-spam "sell" links included in more or less automatically generated text. A very interesting problem for computational linguists and spam-filter designers is then to algorithmically detect a sales pitch in the first case and non-human-generated text in the other.

4.1.5 The search engine APIs

By using the provided APIs we avoid two things: 1) having to parse the interface-pages, which are subject to change at any time, and 2) potentially being blocked from using the search engine due to excessive use.

A third reason to provide the APIs is a direct benefit to the search engine companies: statistics for pages that ultimately are not served to advertisement-reading users are separated from those that are in fact seen by the advertisers' potential customers.

4.1.5.1 Query limits

The drawback with the APIs is that there is only a limited amount of queries available at any time. As defined earlier, one "search" is understood as a single page of results, so checking ten pages of results for a single query-string counts as ten queries, not one. When doing comparative searches it is therefore necessary to keep track of how many queries have already been used up and how many are left. It also means that it pays to look for methods that only utilize the first page of results.

As of this writing, the Google-API had a limit of 1000 queries per developer id per undefined day, that is: the amount of queries available is per person. The limits for Yahoo! Search were 5000 queries per application id per IP address per 24 hour period[14], that is: per IP address the application connects from. An exhaustive search for a translation of *gårdshus*$\diamondsuit$ costs about

[14]When evaluating the compound translator only Yahoo! Search was used for this very reason.

200 queries, meaning there's a very limited amount of exhaustive searches possible per "day".

4.1.5.2 Using the APIs

Google's web search API is based on the *Simple Object Access Protocol* (SOAP) (Gudgin et al., 2003) and *Web Service Definition Language* (WSDL) (Christensen et al., 2001). This is a traditional remote procedure call (RPC) system: a SOAP-client program connects to a SOAP-server, sends the name of a function and the values of its parameters to be run. The server runs the function and sends the results back to the client, which parses the results and if necessary replies with a new set of function/parameters pairs. All the messages are encoded in XML. In the case of the Google API and many other SOAP systems, the allowed nodes, attributes and types of #PCDATA of the XML are specified in a WSDL document.

Yahoo! Search's web search API is based on the REST (Representational State Transfer) (Fielding, 2000, chapter 6) architecture:

> "Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use."

Or in other words, the user interacts with the application by following links in ordinary HTML, XHTML or XML pages served by an ordinary web server using ordinary http.

The fact that Google and Yahoo! Search uses different APIs based on different philosophies makes it necessary to build a common interface to them both if one is to compare and use them both easily. A generalized library I made for this purpose is included in appendix B on page 113.

4.1.6 Testing search engines

Since both the index (the selection of documents) and the query syntax is subject to change, it is necessary to test the search engine to ensure that future results are comparable with previous results. Exactly what needs to be tested depends on what the search engine will be used for, but the testing itself can be as easy as manually testing a set of standard queries, as exemplified in subsection 4.1.2.3 on page 65, which is used to test the hyphen operator.

If the found frequencies increase for all the queries, it is most likely that the index has been enlarged. Testing for this is necessary since Google and Yahoo! Search no longer show an estimate of how many documents they index on their homepages.

If the frequencies change for just a subset of the queries it is possible that there's a been a syntax change. For instance if Yahoo! Search were to implement the hyphen operator the frequency of `page-scraper` at Yahoo! Search would most likely be of the same order of magnitude as the frequency at Google. One cannot assume that such changes will be discussed on the help-pages for the search engine in question. Sometimes new syntax or new operators are added, this of course merits tests to learn what now is possible.

4.2 `ReCompounder`: A prototype compound translator

The entire recompounding process is shown in figure 4.1 on page 81.

At point a), after having been handed an already lemmatized word, `ReCompounder` first tries to look up the word in its bilingual dictionary. If the word is not found it tries to treat the source-language term as a compound by attempting to split the potential compound into stems. If this is possible, each stem is translated into the target language at point b). The translated stems are then recombined first with each other in point c), then with the templates of point d) into possible compounds of the target language at point e). Finally, `ReCompounder` checks whether these potential compounds

are in use on the web at point f), and from the result of that test, in point g), the most frequent candidate is chosen as the best translation.

The assumptions are as follows:

- The stems in a compound each carry their most frequent meanings.
- The translations/meanings in the bilingual dictionary, which currently is the ENE, are sorted by frequency, most frequent first.
- The translations of previously seen compounds are added to the dictionary for faster translation.
- It is sufficient to split a compound only once, into just two stems, because:
  - If a compound stem is used in a larger compound, it must be frequent and have been used before, and as any other stem in a compound it carries only its most frequent meaning.
  - Compounds containing compounds are rare[15] and the more stems per compound, the more likely is the translation to be a rephrasing.

  The `ReCompounder` does not recurse.
- If a compound exists in Norwegian there is a construction in English with approximately the same meaning that already exists on the indexed part of the web.

To illustrate the third point: for the compound *gårdshustak*♢, `ReCompounder` will attempt to find translations for *gårdshus*♢+*tak* and *gård*+*hustak*♢ but not *gård*+*hus*+*tak*, assuming that the last construction is rare to nonexistent.

[15]Non-linguist test-readers of this thesis have pointed out that examples like *gårdshustaksteinsproblem*♢ from chapter 3 are "bad language" that are mainly used by linguists and bureaucrats.

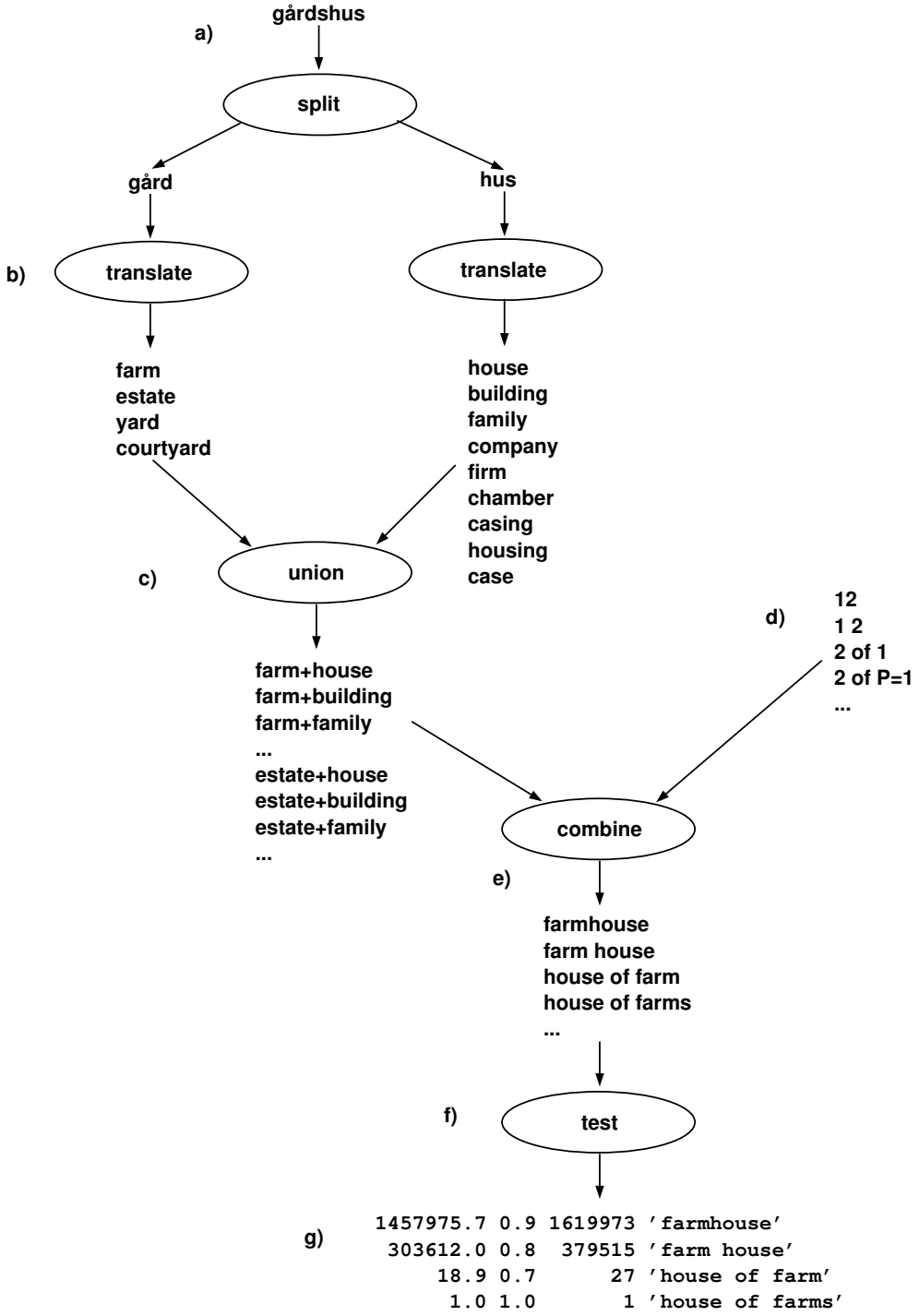


Figure 4.1: The recompounding process from a), the splitting of the compound, to g), the result from the web search.

There is of course no guarantee that the most frequent candidate translation is the *best* translation, especially since the suitability of a translation depends on its intended use.

4.2.1 Use of the dictionaries

The prototype searches through a digitized version of the ENE dictionary, treating it as a treebank, as described in section 2.2 on page 9. Furthermore, the number of known Norwegian stems have been increased by the stems in the NorKompLeks-lists, and checked against Bokmålsordboka.

4.2.2 The compound splitter/recognizer

While it is possible to bypass the compound splitter entirely, it was necessary to build one because when this project was started, the LOGON-grammar for Norwegian did not analyze the compounds beyond POS-tagging them. The compound splitter therefore does not have as many rules as as the one used in Oslo-analysatoren as described in 3.2.1.1 on page 46.

As it is, the compound splitter[16] at point a), figure 4.1 on the preceding page, works by attempting to split the word in two. Each piece, hereafter called the *original stems*, is then looked up in a list of known stems, so as to filter out misspelled stems.

4.2.3 The stem-translator

After having been split, each original stem is looked up in the bilingual dictionary at point b). The translations of each are then stored in a single ordered, duplicate-free list per original stem, as in example 26.

(26) gård: $\{farm, estate, \ldots\}$
hus: $\{house, building, \ldots\}$

These are called the *stem candidates*.

[16]The compound splitter can also be used as a compound recognizer

4.2.4 Potential translation candidates

The stem candidate*s* are then combined in point c): each of the stems in the first list are combined with each of the stems in the second list, preserving order as seen in example 27.

(27) {{farm, house}, {farm, building}, ...
{estate, house}, {estate, building}, ...}

The *translation candidates* are the strings to be tested with a web search engine. They are derived by combining the above pairs of stem candidates with the *templates* from point d), resulting in the translation candidates at point e).

In table 4.2, the examples 16 to 18 on page 48 have been used to show how the templates bridge the translation between the Norwegian and English words.

Norwegian	Template	English
$gård_1$+hus_2	12	$farm_1 house_2$
by_1+$gård_2$	1 2	$apartment_1$ $building_2$
$forretning_1$+$strøk_2$	A=1 2	$commercial_1$ $area_2$
$produksjon_1$+$middel_2$	P=2 of 1	$means_2$ of $production_1$
$streik_1$+$rett_2$	2 to 1	$right_2$ to $strike_1$
$vinter_1$+dag_2	G=1 2	$vinter_1$'s day_2

Table 4.2: Translation templates. The left column show the lemmatized Norwegian compounds. The middle column show the translation templates from point d) in figure 4.1 on page 81. The right column show the result after replacing the digits of the template with the translation of the stems in the left column, at point e) in the recompounding process. The first stem in the source compound is symbolized with the digit 1, the second with the digit 2[18]. The predefined templates can add whitespace to the final translation, as with template **1 2**, reorder the target stems as in template **2 to 1** or mark that a stem needs the be morphologically changed in some way as in template **A=1 2**. A= stands for adjectivization, P= for pluralization and G= for the addition of the genitive s. Adjectivization is only theoretical at this point as a better mapping from noun to adjective is needed, the experimental derivator overgenerates too much to be really useful.

[18]The upper limit of the number of stems in the template-system is 9.

4.2.5 Testing the candidates

Each translation candidate is then tested by doing a web search[19] on Google or Yahoo! Search, at point f), yielding a list like the one at point g). The first column is the frequency adjusted for quality, the second is the quality, the third is the raw frequency as extracted from the search and the fourth is the translation candidate.

Adjusting for quality is needed due to the limits summed up at the end of section 4.1.2.1 on page 65. `ReCompounder` checks the hit context of the first ten hits to ascertain how many of them actually contain the wanted word or phrase, without intervening punctuation. This becomes the *quality* of the search, a number between 0.0 (the query wasn't found in any of the ten hits) to 1.0 (the query was found in all ten hits). Multiplied with the frequency one arrives at the *adjusted frequency*. I here assume that following pages of results have the same ratio of good hits to bad[20]. This of course does *not* protect against cases as in line 54 in listing 4.1 on page 61, where we have a verb and its object and not a compound. The ctq-measure of equation 3.1 on page 53 that was discussed in section 3.3.3 on page 51 was not used because this would increase the cost (number of searches per translation candidate) so much that it would only be possible to process a few compounds per day, making it much less suitable for an actual machine translation system.

4.2.6 Evaluation

The prototype was tested on the evaluation corpus that was described in section 2.3.2 on page 33. It is a collection of tourist information built from web pages that had both a Norwegian and an English version. Stylistically they resemble advertisements, having a high frequency of adjectives per sentence.

The evaluation corpus consisted of 112 sentences including fragments. A

[19]The system as it stands does no domain checking what so ever, using the entire index for its searches.

[20]The number of bad hits generally increases, but does so depending both on filtering done by the particular search engine as well as the number of hits, so a better adjustment would be both query- and search engine-dependent.

	Count	%of nouns
Nouns, total	228	100.0%
Known	151	66.2%
Unknown, not recompoundable	20	8.8%
Unknown, recompoundable	57	25.0%

Table 4.3: Overview of the nouns in the corpus.

locally made POS-tagger was used to extract the common nouns, finding 263 potential, inflected, nouns. While the tagger is capable of returning uninflected forms of known words, compounds are generally unknown, so a basic lemmatizer based on the same principle as the compound splitter was made to derive the base forms of the nouns.

The lemmatizer rejected 21 of the tagged nouns as they did not end with an inflected noun or noun stem[21], another 14 words mistagged as nouns was found by manual inspection. Of the remaining, 151 already had translations in the dictionary and was discarded. At this point, duplicates were removed, leaving 77 potential compounds to be translated by `ReCompounder`.

`ReCompounder` had translation suggestions for 57 of the remaining words.

While most of the failures were due to misspellings and mistaggings that had not been caught earlier or the original stems not being in the bilingual dictionary, there were two very interesting instances of a related, non-software problem. The word *bautastein*♢ could not be translated because the original stem *bauta* translates to "menhir" (a menhir is a standing stone), and there were no instances of `menhir stone` or with other synonyms for stone on the web. This is a very strong indication that the word *bautastein*♢ itself needs to be added to the bilingual dictionary. A similar case was *fiskevær*♢, meaning 'fishing village'. One of the meanings of the stem *vær* is also 'fishing village', ergo one wound up with a reduplicated stem in the candidate.

Evaluation was done by having a bilingual human translator check the suggestions. For each list of translation candidates, the evaluator was presented with a context for the original compound and with up to five of the best candidates, as seen in figure 4.2 on page 87. The evaluator was then

[21]and thus not translatable in the current system

Compound	Type of error
bautastein♢	Synonym of shorter term, *bauta*. Needs own entry in dictionary
bob-bane♢	The borrowed stem "bob" is unique to this compound. Add either bob or bob-bane to dictionary
busk	Not a compound. Improve pre-filtering
busstasjon♢	Split-error, splits as "buss"+"stasjon". Add more rules to the splitter
drømmehytte	"drømme" is either a verb or an e-epenthesized noun "drøm". Also translate V+N-compounds or improve splitter
fiskevær♢	Synonym of shorter term, *vær*. Needs own entry in dictionary
friluftsaktivitet♢	"friluft" 'open air' was not in the dictionary
go-kart♢	Misspelling of *gokart*. Borrowed from English *go-cart*
høyfjells-følelse♢	'høyfjells' is not in the dictionary, unnecessary hyphen
kafe	Not a compound. Improve pre-filtering
**kunsthåndtverker*	Misspelling of *kunsthåndverker*
luksuriøse♢	Not a noun, not lemmatized, wrongly POS-tagged. Improve tagger
overnattingsmulighet♢	"overnatting" is missing from dictionary
sjørøye♢	Couldn't find "røye", suspected charset problem
sjøørret♢	Couldn't find "ørret", suspected charset problem
**souvenir*	Misspelling of the non-compound *suvenir*
storhytte♢	Adjective-noun, outside the scope of the program
storvilt♢	Adjective-noun, outside the scope of the program
sørside♢	Adjective-noun, outside the scope of the program
turistveg♢	*veg*♢ is a spelling variant of *vei*, and refers to the latter

Table 4.4: Words that failed to translate. Several of the cases would be translatable given spell-checked source words or a compound-splitter with more rules.

```
42. 604 meter over Lysefjorden henger denne markante fjellformasjonen.

fjellformasjon

[ ] mountain formation
[ ] formation of mountains
[ ] formation of mountain
[ ] formation of mount
[ ] mount formation
```

Figure 4.2: Task 42 for the evaluator. Each box can be marked with either **Y** for a good translation, **N** for a definitely bad translation or **?** for a possible but not very good translation.

asked to quickly mark each candidate as to whether it was an acceptable translation (with **Y**), completely unacceptable (with **N**) or potentially acceptable (with **?**).

	Ranked	Random
Highest ranked is ok	36.9%	8.6%
At least one, not the top, is ok	31.6%	19.0%
None ok but at least one maybe	17.5%	22.4%
At least one useable in top 5	**86.0%**	**50.0%**
No good candidates	14.0%	50.0%

Table 4.5: Evaluation-results: five best existing collocations sorted by score, then a random sample of size 5 from all possible combinations. The line in boldface is a summary of the rows above.

As can be seen from table 4.5, the evaluator agreed with `ReCompounder`'s top pick more than one third of the time. More interestingly, there was at least one good translation out of five about two thirds of the time. When comparing this to Grefenstette's results from 1999 (discussed in section 3.3.2 on page 50), of 87% correct translations from German to English, it is important to keep in mind the following: 1) Grefenstette only attempted to translate *compounds with a known good translation* while the ReCompounder only attempts to translate *previously unseen compounds*, 2) Grefenstette used a single template (**N N**), while the ReCompounder as evaluated used 5 (**NN**,

N N, **N of N**, **N of plural(N)**, **N by N**)[22].

Table 4.6: Data from the evaluation rounds. The left column is the original Norwegian compound, the middle column has five possible translations selected by random, the right column shows the five best possible translations as ranked by the quality-adjusted search engine frequency. The letter between brackets shows the evaluator's judgment.

Norwegian	English, random	English, ranked
aldersgruppe	[Y] agegroup	[Y] age group
	[N] periodgroup	[Y] agegroup
	[N] group by period	[N] group of age
	[N] group of period	[N] group by age
	[N] group of ages	[N] group of ages
ankomststed	[N] spot by advent	[Y] place of arrival
	[N] place of comings	[?] arrival location
	[N] place of coming	[N] coming place
	[N] cominglocality	[N] place of coming
	[N] arrivallocality	[N] coming location
boenhet	[?] house unit	[?] home unit
	[N] piece by estate	[N] piece of home
	[N] unit of homes	[N] estate unit
	[N] abode unity	[?] home piece
	[N] house unity	[N] estate piece
bondegårdsferie	[?] vacation by farm	[Y] farm vacation
	[Y] farmvacation	[?] farmholiday
	[Y] farmholiday	[N] farm break
	[?] holiday by farm	[?] farmvacation
	[N] holiday of farms	[N] break of farms
buss-selskap	[Y] buscompany	[Y] bus company
	[N] society of buses	[?] coach company
	[N] dowelcompanionship	[N] bus association

Continued on next page

[22]Since the evaluation, the templates **plural(N) of N** and **N's N** have been added

Norwegian	English, random	English, ranked
	[N] corporation of coachs	[N] bus party
	[N] treenail company	[N] bus society
buss-stasjon	[N] treenail depot	[N] bus depot
	[N] depot of dowel	[?] coach station
	[N] station by treenail	[Y] busstation
	[N] depot by treenail	[N] station by bus
	[N] quidstation	[N] coach depot
campinghytte	[N] chalet by camping	[Y] camping cabin
	[N] caravanningchalet	[Y] camping hut
	[N] caravanningcabin	[?] campingcottage
	[N] caravanning hut	[?] campingcabin
	[?] campingchalet	[?] campinghut
campingplass	[N] caravanning square	[?] camping spot
	[N] spot of caravannings	[Y] camping place
	[N] caravanningpost	[N] camping space
	[N] job by camping	[N] camping situation
	[N] caravanningplace	[?] campingplace
caravanplass	[N] seat of trailer	[N] trailer space
	[N] post of trailers	[Y] caravan accommodation
	[N] position of caravans	[N] trailer spot
	[N] farm of caravans	[N] trailer post
	[?] caravansquare	[?] caravan place
dagskort	[N] card of birthday	[?] daycard
	[N] card of nowadayses	[N] card of day
	[N] card by daylight	[N] card of birthday
	[N] card of weekdays	[N] weekday card
	[N] card of daylight	[N] card by day
dalbane	[N] dingle pitch	[N] glenfield
	[N] vale railroad	[N] valley railway

Continued on next page

Norwegian	English, random	English, ranked
	[N] course of valleys	[N] valleyfield
	[N] death of dells	[N] valley railroad
	[N] orbit of dales	[N] valeweb
dansetilbud	[N] proposition of balls	[N] dance supply
	[N] estimate of dance	[N] ball offer
	[N] ballquotation	[N] ball sale
	[N] supply by ball	[Y] dance facility
	[N] dance tender	[N] ball supply
elitedivisjon	[N] league by flower	[N] elite league
	[N] flowerleague	[Y] elite division
	[N] league of picks	[N] elitedivision
	[N] league of flowers	[N] flower division
	[Y] elite division	[N] division of elite
ferieform	[N] shape by holiday	[?] type of vacation
	[N] holiday ending	[Y] type of holiday
	[N] variety by break	[Y] kind of holiday
	[N] holidayending	[Y] variety of holiday
	[N] state by holiday	[?] variety of vacation
feriehus	[?] chamber of holiday	[Y] holiday house
	[N] holidaycasing	[?] vacation house
	[?] holiday chamber	[N] holiday company
	[N] vacation firm	[N] vacation company
	[N] housing by break	[?] holidayhouse
fiskemulighet	[N] angling risk	[N] risk of fish
	[N] anglingpotentiality	[N] potential of fish
	[N] anglingoption	[N] possibility of fish
	[N] fisheries potential	[N] chance of fish
	[N] potential by angling	[?] fish opportunity
fiskeslag	[N] tack of fishing	[N] fishing club

Continued on next page

Norwegian	English, random	English, ranked
	[N] battle by fishing	[Y] kind of fish
	[N] beat by fisheries	[N] angling club
	[N] round of anglings	[N] fishing team
	[N] fisheriesgame	[N] fishing party
fiskevær	[N] weather of fishing	[N] smell of fish
	[N] fishing storm	[N] fish smell
	[N] angling weather	[N] fish breath
	[N] weather of fisherieses	[N] fish wind
	[N] weather of fisheries	[N] fish weather
fjellformasjon	[Y] mountformation	[Y] mountain formation
	[N] formation by mountain	[?] formation of mountains
	[?] formation of rocks	[N] formation of mountain
	[Y] formation of rock	[N] formation of mount
	[Y] formation of mounts	[N] mount formation
fjellovergang	[N] desertion of mounts	[Y] mountain pass
	[?] changeover by mountain	[N] rock crossing
	[Y] mountcrossing	[N] rock pass
	[N] passage of bens	[N] rock connection
	[N] mountain change	[Y] mountain crossing
fjellvegg	[N] partition by moun	[N] rockwall
	[N] wall of alp	[?] rock wall
	[N] wall of mounts	[?] mount wall
	[N] partition of mountain	[?] wall of rock
	[N] mount partition	[Y] mountain wall
fjordarm	[N] fjord fork	[N] fjordarm
	[N] tentacles of firth	[N] fjord arm
	[Y] fork of fjord	[Y] inlet branch
	[N] firthbranch	[N] inlet lever
	[N] branch by firth	[N] firth branch

Continued on next page

Norwegian	English, random	English, ranked
fossefall	[N] fall of cascades	[N] waterfall drop
	[N] waterfalldip	[N] fall of fall
	[N] dip of cascades	[N] fall wave
	[N] waterfall fall	[N] cascade fall
	[N] cascadedrop	[N] waterfall decline
frokostbord	[Y] breakfasttable	[Y] breakfast table
	[N] plank by breakfast	[N] breakfast board
	[N] breakfast plank	[?] breakfasttable
	[?] table of breakfasts	[N] board of breakfast
	[N] breakfast board	[?] table of breakfast
gangavstand	[N] lapsegap	[Y] walking distance
	[N] interval of steps	[N] time interval
	[N] interval of passages	[N] interval of time
	[N] gap by vein	[N] working distance
	[N] action distance	[N] time gap
gjestegård	[N] estate of customers	[Y] guest farm
	[N] yard by patron	[?] guestfarm
	[?] guestcourtyard	[N] guest estate
	[N] farm of customer	[N] customer farm
	[N] farm by guest	[N] guest yard
golfbane	[N] alley of golf	[?] golf course
	[N] alley of gulf	[Y] golfcourse
	[N] track by gulf	[Y] golflinks
	[N] path of gulfs	[N] golf track
	[N] gulf course	[N] golf field
gravrøys	[N] sepulchreheap	[N] bunch of holes
	[N] heap of grave	[N] hole bunch
	[N] heap of holes	[N] pit heap
	[Y] tomb heap	[N] loads of holes

Continued on next page

Norwegian	English, random	English, ranked
	[N] bunch by grave	[N] bunch of pit
havgap	[N] jaws by sea	[N] ocean opening
	[N] gulf of seas	[N] sea gap
	[N] opening by ocean	[N] opening of ocean
	[?] gape of oceans	[N] opening of sea
	[?] gape of seas	[?] seagulf
havstue	[N] cottage of ocean	[N] patch of ocean
	[?] ocean cottage	[N] patch of sea
	[Y] cottage by sea	[N] sea patch
	[?] oceancottage	[N] sea mound
	[N] cottage of oceans	[N] seapatch
hotellferie	[?] vacation by hotel	[?] hotel break
	[N] holiday of hotels	[Y] hotelholiday
	[?] break by hotel	[?] hotelvacation
	[N] vacation of hotels	[N] vacation of hotels
	[Y] hotelbreak	[Y] holiday by hotel
hyttedør	[?] chalet door	[Y] cabin door
	[N] door by hut	[Y] cottage door
	[N] poop door	[Y] hut door
	[N] door of smeltery	[?] chalet door
	[N] door of poop	[?] cottagedoor
hytteferie	[N] vacation of poop	[?] cottage holiday
	[Y] cabinholiday	[N] chalet holiday
	[N] holiday of poops	[Y] cabin holiday
	[N] vacation by cottage	[N] cottage break
	[?] holiday by cabin	[?] cabin break
høydeforskjell	[?] disparity of altitudes	[Y] height difference
	[N] distinction of hill	[?] level difference
	[N] staturedifference	[?] elevation difference

Continued on next page

Norwegian	English, random	English, ranked
	[N] tallness disparity	[Y] altitude difference
	[N] difference of perpendicular	[N] level distinction
kveldstid	[N] eveningage	[?] nighttime
	[N] nightday	[N] time of night
	[N] tense of nights	[Y] evening time
	[N] eveningday	[N] night season
	[N] night term	[Y] evening hour
luksusferie	[N] vacation by luxury	[Y] luxury holiday
	[?] holiday of luxuries	[?] luxury break
	[?] break of luxury	[?] luxuryvacation
	[N] holiday by luxury	[?] luxuryholiday
	[?] vacation of luxuries	[N] vacation of luxury
mattilbud	[?] sale of grub	[N] food supply
	[N] bargain by food	[N] supply of food
	[N] faresale	[N] fare sale
	[N] tender of cuisine	[?] sale of food
	[N] quotation of foods	[?] food facility
musikktilbud	[N] proposition by music	[N] group offer
	[N] offer by orchestra	[N] music sale
	[N] tender of band	[N] group facility
	[N] orchestratender	[N] group bid
	[N] proposition by group	[?] music offer
naturopplevelse	[N] temperament thrill	[Y] nature experience
	[N] adventure of geographies	[?] experience of nature
	[N] temperamentthrill	[N] perception of nature
	[N] sceneryadventure	[N] landscape experience
	[N] treat of temperament	[N] feeling of nature
opplevelsespark	[N] spark of perception	[Y] adventure park
	[N] perception spark	[N] adventure playground

Continued on next page

Norwegian	English, random	English, ranked
	[N] kicksled of feelings	[N] treat common
	[N] crossled of treat	[?] thrill park
	[N] punt of treats	[N] experience park
parkeringsplass	[N] rank of parking	[Y] parking space
	[N] parking room	[N] parking spot
	[?] parking place	[N] parking place
	[N] job of parkings	[N] parking situation
	[N] place of parkings	[N] parking position
reinsdyrskinn	[N] caribouappearance	[Y] reindeer skin
	[Y] reindeerhide	[Y] reindeer hide
	[N] reindeer light	[Y] reindeer fur
	[N] glow by caribou	[?] reindeer leather
	[N] cariboufur	[N] reindeer light
rorbu	[N] tillerleanto	[N] wheel shed
	[?] rudder guardhouse	[N] wheel shack
	[N] shack of rudders	[?] shed by helm
	[N] guardhouse of elevators	[N] helm shack
	[N] fantail glasshouse	
ruteopplysning	[N] servicelearning	[N] square lighting
	[?] schedule enlightenment	[N] square learning
	[N] glassenlightenment	[N] pane information
	[N] glasslighting	[N] square education
	[N] service teaching	[N] knowledge of square
sentrumsområde	[N] campus of downtowns	[N] centerfield
	[N] downtown department	[N] center area
	[N] domain by centre	[N] downtown campus
	[?] centresector	[?] district of downtown
	[N] province by downtown	[Y] centre region
serveringtilbud	[N] tender by service	[N] service facility

Continued on next page

Norwegian	English, random	English, ranked
	[N] offer of services	[N] service offer
	[N] serviceproposition	[N] supply of services
	[N] tender of services	[N] sale of services
	[N] servicesupply	[N] offer of services
sjokoladefabrikk	[N] chocolatemill	[Y] chocolate factory
	[N] chocolate mill	[?] chocolatefactory
	[N] mill by chocolate	[N] chocolate mill
	[N] factory of chocolate	[N] factory of chocolate
	[Y] chocolatefactory	[N] chocolatemill
sjøhus	[N] swellfirm	[N] lake house
	[?] chamber by lake	[N] lakehouse
	[N] swellbuilding	[N] lake family
	[N] wave company	[N] ocean house
	[N] casing of seas	[?] sea house
sjøkant	[N] arc of lakes	[N] lakeside
	[N] ocean area	[Y] seaside
	[N] arc of lake	[Y] oceanside
	[N] sea part	[?] sea side
	[N] area of swells	[N] sea region
skianlegg	[N] propensity of skis	[N] skid system
	[N] bent of ski	[?] ski system
	[N] diathesis of ski	[N] ski works
	[N] skid proneness	[N] ski factory
	[N] proclivity of ski	[N] ski talent
skieventyr	[N] dream of skis	[Y] ski adventure
	[N] dream of ski	[N] ski dream
	[N] affair of ski	[N] ski sensation
	[N] affair of skid	[N] skidream
	[N] skid adventure	[N] ski wonder

Continued on next page

Norwegian	English, random	English, ranked
spisested	[?] foodspot	[?] food place
	[N] spot by food	[N] food location
	[N] spot of food	[N] passage of food
	[N] foodlocality	[?] food spot
	[N] passage of foods	[N] location of food
sykkeltur	[?] trip by cycle	[N] bike tour
	[?] excursion by bicycle	[N] bicycle tour
	[N] bicycle period	[Y] bike trip
	[N] luck of bikes	[Y] bicycle trip
	[Y] bicycle excursion	[N] biketour
tregjenstand	[N] wood topic	[?] tree object
	[N] tree target	[Y] wood thing
	[?] treeobject	[Y] wood object
	[N] target of trees	[Y] subject of wood
	[N] subject of trees	[Y] wood target
turistinformasjon	[?] touristinfo	[N] tourist data
	[N] tourist poop	[N] knowledge of tourist
	[N] information by tourist	[N] information of tourist
	[N] gen of tourists	[N] tourist knowledge
	[Y] touristinformation	[N] information of tourists
turmulighet	[?] touroption	[?] trip option
	[N] potential of spells	[Y] trip opportunity
	[?] tour chance	[N] tour opportunity
	[N] possibility of period	[N] tour potential
	[N] figure opportunity	[N] trip risk
weekend-pris	[?] cost of weekends	[Y] weekend rate
	[?] rate of weekends	[Y] weekend price
	[N] weekend award	[N] weekend cost
	[N] rate by weekend	[N] weekend fare

Continued on next page

Norwegian	English, random	English, ranked
	[N] charge by weekend	[N] weekend award

Starts on page 88

Table 4.6 lists all the compounds that were recompounded during the evaluation. The leftmost column are of the source words. The middle column is the baseline[23], produced by doing all the recompounding steps *except* checking the candidates on the web, and instead selecting five candidates at random. The rightmost column lists the properly recompounded nouns.

4.2.7 Consequences for machine translation and lexicography

Already while testing the system during the development stage, certain tendencies emerged. All the compounds so far tested can be categorized as follows:

4.2.7.1 Good to excellent translations

(28) aldersgruppe → alder + gruppe
⟹ age group

The top five of the complete results for *age group* are in table 4.7.

2746668.0	0.8	3433335	age group
11229.4	0.7	16042	agegroup
1451.0	1.0	1451	group of age
747.6	0.7	1068	group by age
295.0	1.0	295	group of ages

Table 4.7: Top five results for “age group”.

4.2.7.2 Source word is not a compound

That a word *can* be split doesn't mean that it is a compound. For instance: while the Norwegian word *habil* can be split into *ha* ‘have’ + *bil* ‘car, au-

[23] . . . and it definitely has some entertainment value, if little else.

tomobile...', it is a borrowing from Latin *habilis* meaning 'competent, able, qualified'.[24]

4.2.7.3 Source word only makes sense in a bigger structure

This is the case with *dalbane*♢ on page 89. It is the last part of the MWE *berg- og dalbane*[25] 'rollercoaster', which does exist in the dictionary but was not detected by the compound-filter.

4.2.7.4 Unsplittable compound

No stems to search, no result. Often this is due to a compound-only suppletive stem missing from the dictionary or wordlists, as discussed in section 3.2, example 7b on page 42.

4.2.7.5 Bilingual dictionary lacks usable English stem

Sometimes, this is due to the stem missing from the dictionary, but there are also cases where there are no single word translations of the stem.

(29) seterbu → seter + bu
⟹ no useful translations + shack, hut, cabin...

In the above case, there are no suitable one-word translations of *seter*. In this case the target language lacks an adquate term, while in the last category it is the source language that lacks an entry for the adequate meaning.

[24]Several words ending with *-bil* comes from Latin words ending with *-bilis*, like *stabil* from *stabilis* and *mobil* from *mobilis*.

[25]This word probably derives from **bergbane og dalbane*. This is a general technique in Norwegian: when two compounds share the last stem and are conjoined, the first redundant stem is removed. Another example is *fiskeri- og landbruksdepartementet*♢ 'the Ministry for Fishing and Agriculture'.

4.2.7.6 One of the stems of the compound means the same as the compound

This was the case with *bautastein* and *fiskevær* as discussed in section 4.2.6 on page 84, and is the case with *fossefall* on page 92. To recapitulate: some compounds are as a whole synonymous with one of its stems. A *bautastein*$\diamondsuit$ is a *bauta* and a *fiskevær*$\diamondsuit$ is a *vær*, and a stem by stem translation might end up duplicating one of the stems.

4.2.7.7 The meaning of the Norwegian compound stem systematically differs from standalone stem

(30) ulvestamme $\rightarrow$ ulv + stamme
$\Longrightarrow$ wolf + no useful translations

This might be the most interesting result to lexicographers. While both *ulv* 'wolf' and *stamme* 'trunk, bole, stock, tribe, clan, core, nucleus, stem, root, mandrel, strain, stock, game population' exist in the ENE dictionary, the last stem *stamme* is synonymous with *bestand* 'population' when used in a compound where the first stem denotes a mammal. An *ulvestamme* is approximately 'total population of wolves in a certain area'. The `ReCompounder`, using shallow processing at all points, cannot massage any of the recorded translations into a synonym for *population*.[26]

A similar problem is *bil* 'car, automobile, motor, auto, taxi, cab, taxicab'. When in a compound it generally translates to 'truck'[27], as looking at the compounds ending with *bil* in the ENE dictionary will show. In these compounds, *bil* is translated with 'car', 'truck' or 'lorry' unless the English equivalent is a specialized word, e.g. *likbil* 'hearse', *lastebil* 'truck', *rånebil* 'hot rod' or *sykebil* 'ambulance'.

[26] The best recorded translation is 'game population', which cannot be used in any of the templates directly since it consists of two words. It would take world knowledge to know that wolves might be considered a kind of game and therefore delete the string `"game"` from the translation before filling the template.

[27] Norw. *lastebil*, *trailer*.

`ReCompounder` can be used part of the way to discover such holes in the dictionary: by translating several compounds sharing a stem and comparing the translations, akin to the mirrors-method (Dyvik, 1998): translating to a second language and back, effectively using the second language as a filtering mechanism. Making the definition of the missing stem or meaning would still fall to a lexicographer. The ENE dictionary already notes expressions containing the head word. Perhaps it is time to also record specialized meanings that occur in compounds.

4.2.8 Consequences for manual translation

Both the ideas behind the `ReCompounder` and using web search directly are useful for manual human translation. While checking all possible recompounds of a word by hand is a repetitive and therefore error-prone task, a human can select a subset of compounds that are deemed most likely and check only those. Web search is also useful for other quick checks on intuition, for instance which prepositions cluster with which verbs[28].

4.3 Future work

If we are to use search engines directly for computational linguistics we'll need better algorithms to decrease the returned frequency to better reflect a recovery search. In essence we need to improve precision after the fact. The `ReCompounder` lowers frequency depending on the quality of the first ten hits: if one hit is irrelevant, the frequency is lowered by 10%, if two hits are bad by 20% etc., however as studying all returned hits with frequencies lower than 1000 shows, a linear decrease is insufficient. This ties in with the need for third party evaluation of search engines, whether it is to be used for linguistics or other data mining by third parties. Another approach is to improve precision by filtering out noise in the search engine itself, which practically limits such research to the search engine companies, or while searching, which is relevant for any search engine user.

[28]For instance cater to/cater for

As for translating compounds and MWEs, the main challenge is to find more templates, and to find templates for longer compounds. The latter will eventually yield diminishing returns as the likelihood of rephrasing goes up with the length of the compound. Another aspect is to improve and add to the derivational and inflectional machinery for the existing templates, for instance by the addition of a good noun-to-adjective mapper.

Reversing the process by translating from English to Norwegian is hampered by three things: there are fewer texts in Norwegian, all the big search engines were originally tailored for the relatively isolating English and are less suited for more inflecting languages like Norwegian, and while translating a compound into a compound is straightforward, finding and deciding which English non-compound NPs ought to be translated into a compound is hard.

Chapter 5

"Any technology distinguishable from magic is insufficiently advanced."

Benford's corollary to Clarke's third law, 1997

Conclusion

The current quality of machine translated text testifies that there is still much to do before machine translation appears to work like magic, by producing consistently good translations quickly and with no human intervention. To paraphrase the quote we started with: the problem is still that a text must be minimally understood before it can be translated properly, and the challenge is still how to make a computer *understand* something as opposed to *processing* it.

After introducing the problem in chapter 1 on page 1 we looked at the setting and background for this thesis, the machine translation project LOGON, in chapter 2 on page 5. The remainder of that chapter contained descriptions of the linguistic resources that was used for the experimental part of this thesis, all of them made available through LOGON. The most important of these linguistic resources is the bilingual dictionary (Eek et al., 2001) and how to access its data by converting the original XML to s-expressions and using `tgrep2` for the actual searches. Thanks to `tgrep2` we could therefore get away with just a very small introduction to the large and still growing world of XML. Furthermore this also serves as a general hint on how to access other data having a tree-structure: convert it to a different tree-structure preserving format that already has good tools of access.

Compounding in Norwegian, what Norwegian compounds translate to in English, and previous work of automated compound translation was the

topic of chapter 3 on page 37. Of most importance here was the *templates*, mapping Norwegian noun-noun constructions to their equivalents in English.

Chapter 4 on page 57 served as the core of the thesis. Its first part showed what current web search engines are capable of, how they work, how they differ from a static corpus, and how to work around the noise that follows from anyone with access being able to add data. This section also pointed out two different search strategies, *exploratory search* and *recovery search*, and that while search engines catering to casual users support exploratory search, using the web as corpus demands recovery search.

The next part of chapter 4 was practical, showcasing the program `ReCompounder`, a compound translator using the search engines of today to rank translation candidates instead of using a static corpus. This showed that it is still relevant to use current Internet search engines to translate compounds, even though these are based on popularity measures and not pure inverse indexes as was the case of Grefenstette's pilot experiments with AltaVista. Considering the scope of translation and number of templates used by `ReCompounder` I'll go as far as to claim that the results as found in this thesis are both better and of greater interest to the linguistic community than Grefenstette (1999). Compared to a baseline where all combinations are equally likely, the results are pretty good. As a bonus, the system as-is detects errors and gaps in the bilingual dictionary and is therefore useful as a tool for lexicography. The strategy itself will be used in the machine-translation project LOGON to improve coverage there.

Appendix A

XML

A.1 Tools for XML

All the listings in this appendix have been indented for readability and provided with line numbers for the convenience of the reader.

A.1.1 The dictionary after direct conversion to s-expressions

```
1  (dictionary
2
     (entry
4      (@
         (number 1))
6      (word
         (LEAF abacus))
8      (pos
         (@
10         (grammarpage nouns))
         (LEAF n))
12     (definition
         (LEAF ancient manual calculator))
14     (seealso
         (LEAF slide rule)))
16
```

```
(entry
  (@
    (number 2))
  (word
    (LEAF abash))
  (pos
    (@
      (grammarpage verbs))
    (LEAF vt))
  (definition
    (LEAF to make so. ashamed or embarrased))
  (expression
    (words
      (LEAF to abash so. by sneering))))

(entry
  (@
    (number 3))
  (word
    (LEAF abeyance))
  (pos
    (@
      (grammarpage expressions))
    (LEAF n))
  (definition
    (LEAF in expressions only))
  (expression
    (words
      (LEAF
        (property that is)
        in abeyance))
    (meaning
      (LEAF property without an owner)))
  (expression
    (words
```

```
52          (LEAF hold smth. in abeyance))
53        (meaning
54          (LEAF temporarily suspend smth.)))))
```

Listing A.1: The results of converting the example XML without adapting the stylesheet.

A.1.2 A stylesheet tailored for the dictionary

```
1  <?xml version="1.0"?>
2  <xsl:stylesheet
       version="1.0"
4      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="text"/>
6  <xsl:strip-space elements="*"/>

8  <xsl:template match="text()">
       (LEAF <xsl:value-of select="normalize-space(.)"/>)
10 </xsl:template>

12 <xsl:template match="dictionary//*">
       (<xsl:value-of select="local-name()"/>
14     <xsl:apply-templates/>)
   </xsl:template>
16
   </xsl:stylesheet>
```

Listing A.2: XSLT stylesheet adjusted to the mock-up dictionary

A.1.3 The dictionary as s-expressions after conversion by the adjusted stylesheet

```
1  (entry
2    (word
       (LEAF abacus))
4    (pos
       (LEAF n))
6    (definition
       (LEAF ancient manual calculator))
```

```
8     (seealso
9       (LEAF slide rule)))
10
    (entry
12    (word
        (LEAF abash))
14    (pos
        (LEAF vt))
16    (definition
        (LEAF to make so. ashamed or embarrased))
18    (expression
        (words
20        (LEAF to abash so. by sneering))))

22  (entry
      (word
24      (LEAF abeyance))
      (pos
26      (LEAF n))
      (definition
28      (LEAF in expressions only))
      (expression
30      (words
          (LEAF [property that is] in abeyance))
32      (meaning
          (LEAF property without an owner))
34      )
      (expression
36      (words
          (LEAF hold smth. in abeyance))
38      (meaning
          (LEAF temporarily suspend smth.))))
```

Listing A.3: The final results of converting the example XML, after adapting the stylesheet.

A.2 S-expressions to and from XML

A.2.1 XML to S-expressions

The following XSLT stylesheet will convert any XML to S-expressions, it needs to be run by an XSLT processor.

```
<?xml version="1.0"?>
<xsl:stylesheet
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text"/>
<xsl:strip-space elements="*"/>

<xsl:template match="text()">
    (LEAF <xsl:value-of select="normalize-space(.)"/>)
</xsl:template>

<xsl:template match="*">
    (<xsl:value-of select="local-name()"/>
    <xsl:if test="@*">(@ <xsl:for-each select="@*">
        (<xsl:value-of select="local-name(.)"/><xsl:text> </xsl:text>
        <xsl:value-of select="."/>)</xsl:for-each>)</xsl:if>
    <xsl:apply-templates/>)
</xsl:template>

</xsl:stylesheet>
```

Listing A.4: Generic XSLT-stylesheet to convert from XML to s-expressions. Identical to listing 2.3 on page 19.

A.2.2 A program for converting s-expressions to XML

This short library-with-builtin-example-program needs Python 2.3 or newer to run, get it from `http://www.python.org/`.

For usage, invoke it like so: `python lisp2xml.py -h`.

```
#!/usr/bin/env python
# Author : Hanne Moa
# Date   : 2004-12-10
```

APPENDIX A

```
# LICENSE: public domain

import string, sys
from optparse import OptionParser

class Lisp2XML_FST(object):

    def __init__(self, root='ROOT', empty='EMPTY', readfrom=''):
        self.emptytag = empty
        self.roottag = root
        if readfrom:
            self.readfrom = file(readfrom)
        else:
            self.readfrom = sys.stdin
        self._tags = []
        self._next = None
        self._prev = None

    def is_startparens(self):
        self.is_tag()
        self._tags.append([])

    def is_tag(self):
        prevtag = ''.join(self._tags[-1]) or self.emptytag
        self._tags[-1] = prevtag
        self.write("<%s>" % prevtag)

    def is_endparens(self):
        tag = ''.join(self._tags.pop())
        self.write("</%s>" % tag)

    def write(self, something):
        sys.stdout.write(something)

    def comment(self, char):
        if char == '\n':
            return self._prev
        else:
            return 'comment'

    def starttag(self, char):
        self._prev = 'starttag'
        if char in string.whitespace:
            return 'starttag'
        elif char == '(':
            self._tags.append([])
            return 'intag'
        elif char == '%':
            return 'comment'
        else:
```

```
            return 'error'

    def intag(self, char):
        self._prev = 'intag'
        if char == '(':
            self.is_startparens()
            return 'intag'
        elif char in string.whitespace:
            if not self._tags[-1]:
                return 'intag'
            else:
                self.is_tag()
                return 'indata'
        elif char == '%':
            return 'comment'
        elif char == ')':
            self.is_endparens()
            return 'indata'
        else:
            self._tags[-1].append(char)
            return 'intag'

    def indata(self, char):
        self._prev = 'indata'
        default = 'indata'
        if char == '(':
            self._tags.append([])
            return 'intag'
        elif char == ')':
            self.is_endparens()
            return default
        elif char in string.whitespace:
            self.write(char)
            return default
        elif char == '%':
            return 'comment'
        else:
            self.write(char)
            return default

    def error(self, char):
        return ''

    engine = {
        'comment': comment,
        'starttag': starttag,
        'intag': intag,
        'indata': indata,
        'error': error,
        }
```

```
def convert(root='ROOT', empty='EMPTY', readfrom=''):
    fst = Lisp2XML_FST(root, empty, readfrom)
    engine = fst.engine
    fst.write("<%s>" % fst.roottag)
    for line in fst.readfrom:
        for char in line:
            if not fst._tags:
                next = 'starttag'
                fst.write('\n')
            prev = next
            next = engine[next](fst, char)
            if not next:
                if fst._tags:
                    fst.is_endparens()
                break
    if fst._tags:
        fst.is_endparens()
    fst.write("</%s>\n" % fst.roottag)
    fst.readfrom.close()

if __name__ == '__main__':
    usage = """Usage: %prog [options] < FILE
Translate s-epressions into XML."""
    parser = OptionParser(usage=usage)
    parser.add_option("-f", "--file", metavar='FILE',
        help="convert s-expressions in FILE")
    parser.add_option("-e", "--empty", default='EMPTY',
        help="fallback tag in case of '(' in source")
    parser.add_option("-r", "--root", default='ROOT',
        help="tag to use as root, if none")

    (opts, args) = parser.parse_args()

    convert(root=opts.root, empty=opts.empty, readfrom=opts.file)
```

Listing A.5: Source for converting from s-expressions to XML

Appendix B

A common interface to search engine APIs

The interface library in the following listing was designed to be suffciently generic that it should be easy to extend with future search APIs or even page scrapers. The `license`-methods would not be necessary for a page-scraper. The `count`-method will yield the frequency of a query while the results themselves are stored in the attribute `results` in their native format, since only the generic count-method proved necessary to build for this thesis.

```
from __future__ import division, generators
import sys
import re
import socket
import errno

import google
import SOAPpy

import yahoo
import yahoo.search.webservices
from yahoo.search.webservices import WebSearch as YWebSearch

class WebSearchError(Exception):
    errors = {0: 'Unknown error',
              1: "No more licenses",
              2: "No searches left on this license",
              3: "Websearch server error",
              4: "Malformed query",
              }
```

```
    def __init__(self, errno=0):
        self.errno = errno
        self.errmsg = self.errors.get(errno, 'Unknown error')

    def __str__(self):
        return repr(self.errmsg)

class WebSearch(object): #abstract
    ellipse = r'<b>\.\.\.</b>'
    ellipsestripper = re.compile(ellipse)
    htmlstripper = re.compile(r'</?b>')

    def __init__(self, **kwargs):
        self.results = None

    def setLicense(self):
        raise NotImplemented

    def _makeLicensePool(self):
        raise NotImplemented

    def count(self, qstring, maxresults=10, phrases=False):
        """Uses self.search() to actually interact with the search engine."""
        raise NotImplemented

    def search(self, qstring, maxresults=10, phrases=False):
        """Used by self.count() to actually interact with the search engine."""
        raise NotImplemented

    def _has_phrase(self, result, pattern):
        """Checks if a hit actually contains all the words of the query."""
        raise NotImplemented

    def _convertPattern(text):
        """ _convertPattern(s) -> string

        Convert from google pattern to re-module regexp

        >>> _convertPattern("a * house")
        'a .*? house'
        """
        return re.sub(r'\*', '.*?', text)
    _convertPattern = staticmethod(_convertPattern)

    def phrasequote(astring, quote=True):
        if quote and astring.count(' ') and not (astring[0] == astring[-1] == '"'):
            return '"'+astring+'"'
        else:
            return astring
    phrasequote = staticmethod(phrasequote)
```

APPENDIX B

```
    def _checkresults(self, results, pattern=''):
        """Returns the number of hits from the results that actually
        contain the pattern."""
        if pattern:
            pattern = re.compile(self._convertPattern(pattern))
            return sum([self._has_phrase(result, pattern) for result in results])
        else:
            return 0

class GoogleWebSearch(WebSearch):

    def __init__(self, pathtopool='.licenses', keys={}):
        super(GoogleWebSearch, self).__init__()
        self.pool = self._makeLicensePool(pathtopool)
        try:
            self.setLicense(keys['google'])
        except WebSearchError, val:
            print >>sys.stderr, val
            sys.exit(1)

    def _has_phrase(self, result, pattern):
        title = self.htmlstripper.sub('',
                self.ellipsestripper.sub('', result.title.strip().lower()))
        context = self.htmlstripper.sub('',
                self.ellipsestripper.sub('', result.snippet.strip().lower()))
        if pattern.findall(title) or pattern.findall(context):
            return True
        else:
            return False

    def _makeLicensePool(self, pathtopool='.licenses'):
        """Fetches the next available license, if any
        """
        IF = file(pathtopool)
        lines = IF.readlines()
        IF.close()
        for line in lines:
            yield line.strip().split()[0]

    def _newLicense(self):
        """Changes to the next available license in the pool, if any
        """
        try:
            google.setLicense(self.pool.next())
        except StopIteration:
            print 'No more licenses available, aborting...'
            raise WebSearchError, 2

    def setLicense(self, license=''):
```

```
        """Changes to the next available license in the pool, if any
        """
        try:
            google.setLicense(license)
        except:
            print 'License unavailable or invalid, trying pool...'
            self._newLicense()
            print 'OK'
            raise WebSearchError, 1

    def search(self, qstring, maxresults=10, phrases=False):
        """count(s[, maxresult]) -> data

        Check how many hits 'qstring' gets on google, testing the
        'maxresults' results to see if 'qstring' is actually there.

        Returns the results in search engine API native format.
        """
        try:
            data = google.doGoogleSearch(self.phrasequote(qstring.lower(), phrases))
        # socket.error: (104, 'Connection reset by peer')
        except socket.error, val:
            if val and type(tuple) == type(val) and val[0] == errno.ECONNRESET:
                raise WebSearchError, 0
        except SOAPpy.Types.faultType, val:
            raise WebSearchError, 2
        self.results = data
        return data

    def count(self, qstring, maxresults=10, phrases=False):
        """count(s[, maxresult]) -> (int, float, bool)

        Check how many hits 'qstring' gets on google, testing the
        'maxresults' results to see if 'qstring' is actually there.

        Returns a tuple of:
            int(ghits)
            float(quality) of result, between 0 and 1
            bool(accurate), whether ghits is an estimate or not
        """
        data = self.search(qstring, maxresults, phrases)
        ghits = data.meta.estimatedTotalResultsCount
        accurate = data.meta.estimateIsExact
        results = data.results[:maxresults]
        quality = self._checkresults(results, qstring[1:-1])
        normalized_quality = 0.0
        if quality:
            normalized_quality = quality / maxresults
        return int(ghits), normalized_quality, bool(accurate)
```

APPENDIX B

```
class YahooWebSearch(WebSearch):
    __appid = None

    def __init__(self, pathtopool='.licenses', **unused):
        super(YahooWebSearch, self).__init__()
        self.pool = self._makeLicensePool()
        self.setLicense()
        self._ysearch.results = 10

    def _has_phrase(self, result, pattern):
        title = self.htmlstripper.sub('',
                self.ellipsestripper.sub('', result['Title'].strip().lower()))
        context = self.htmlstripper.sub('',
                self.ellipsestripper.sub('', result['Summary'].strip().lower()))
        if pattern.findall(title) or pattern.findall(context):
            return True
        else:
            return False

    def _makeLicensePool(self, pathtopool='.licenses'):
        """Fetches the next available license, if any
        """
        if self.__appid:
            yield self.__appid
        else:
            IF = file(pathtopool)
            lines = IF.readline() # multiple ids unneccessary for yahoo
            IF.close()
            yield line.strip().split()[0]
            raise (WebSearchError, 1)

    def setLicense(self, unused=''):
        """Changes to the next available license in the pool, if any
        """
        self._ysearch = YWebSearch(self.pool.next())

    def search(self, qstring, maxresults=10, phrases=False):
        """count(s [, maxresult]) -> (int, float, bool)

        Returns the results in search engine API native format.
        """
        try:
            self._ysearch.query = self.phrasequote(qstring.lower(), phrases)
        except:
            print 'Some Error!'
            raise

        try:
            data = self._ysearch.parse_results()
        except yahoo.search.webservices.ServerError, val:
```

```
                if val == "Internal WebService error":
                    raise WebSearchError, 3
                elif val == "WebService server unavailable":
                    raise WebSearchError, 1
                else:
                    raise WebSearchError, 0
            except yahoo.search.webservices.SearchError, val:
                print val
                raise
                raise WebSearchError, 2
            except:
                raise
                raise WebSearchError, 0
            self.results = data
            return data

        def count(self, qstring, maxresults=10, phrases=False):
            """count(s [, maxresult]) -> (int, float, bool)

        Check how many hits 'qstring' gets on google, testing the
        'maxresults' results to see if 'qstring' is actually there.

        Returns a tuple of:
            int(ghits)
            float(quality) of result, between 0 and 1
            bool(accurate), whether ghits is an estimate or not
        """
            data = self.search(qstring, maxresults, phrases)

            maxresults = min(maxresults, data.totalResultsReturned)

            yhits = data.totalResultsAvailable
            results = data.results[:maxresults]
            quality = self._checkresults(results, qstring[1:-1])
            normalized_quality = 0.0
            if quality:
                normalized_quality = quality / maxresults
            return int(yhits), normalized_quality, None

    if __name__ == '__main__':
        crawler = YahooWebSearch()
        teststring = '"the ** chair"'
        a, b, c = crawler.count(teststring)
        if a == b == c == None:
            pass # change to new license and retry
        print repr(teststring), a, b, c
```

Listing B.1: An abstraction layer for the web search APIs of Google and Yahoo! Search, written in Python

Bibliography

Bar-Hillel, Yehoshua (1960). The Present Status of Automatic Translation of Languages. *Advances in Computers*, 1:pp. 91–163.

Bauer, Laurie (2004). Adjectives, compounds and words. *Nordic Journal of English Studies*, pp. 7–22.

Benford, Gregory (1997). *Foundation's fear*. Novel.

Bergenholtz, Henning, Ilse Cantell, Ruth Vatvedt Fjeld, Dag Gundersen, Jón Hilmar Jónnson, and Bo Svensén (1997). *Norsk leksikografisk ordbok*, vol. 4 of *Skrifter utgitt av Nordisk forening for leksikografi*. Universitetsforlaget. ISBN 82-00-22901-7.

Borthen, Kaja (2003). *Norwegian Bare Singulars*. Ph.D. thesis, NTNU, Trondheim.

Bresnan, Joan (2001). *Lexical-Functional Syntax*. Blackwell Textbooks in Linguistics. Blackwell. ISBN 0631209743.

Butt, Miriam, Tracy Holloway King, María-Eugenia Niño, and Frédérique Segond (1999). *A grammar writer's cookbook*. No. 95 in CSLI lecture notes. CSLI Publications, Stanford, CA, USA. ISBN 1575861704.

Cao, Yunbo and Hang Li (2002). Base noun phrase translation using Web data and the EM algorithm. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING 2002)*. Taipei, Taiwan.

Carroll, John and Stephan Oepen (2005). High efficiency realization for a wide-coverage unification grammar. In Dale, Robert and Kam-Fai Wong

(eds.), *Proceedings of the Second International Joint Conference on Natural Language Processing (IJCNLP05)*, vol. 3651 of *Springer Lecture Notes in Artificial Intelligence*, pp. 165–176. Springer-Verlag, Jeju Island, Korea.

Celko, Joe (2004). *Trees and Hierarchies in SQL for Smarties*. Morgan Kaufmann. ISBN 1558609202.

Christensen, Erik, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana (2001). *Web Services Description Language (WSDL) 1.1*. Tech. rep., W3C.

Clark, James et al. (1999). *XSL Transformations (XSLT) Version 1.0*. Tech. rep., W3C.

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):pp. 377–387.

Copestake, Ann (2002). *Implementing Typed Feature Structure Grammars*. No. 110 in CSLI Lecture Notes. CSLI publications, Stanford, CA, USA. ISBN 1-57586-260-3.

Copestake, Ann, Dan Flickinger, Rob Malouf, Susanne Riehemann, and Ivan Sag (1995). Translation using Minimal Recursion Semantics. In *Proceedings of the 6th. International Conference on Theoretical and Methodological Issues in Machine Translation (TMI-95)*. Leuven, Belgium.

Corbett, Greville (1991). *Gender*. Cambridge textbooks in linguistics. Cambridge University Press. ISBN 0-521-33845.

Dalrymple, Mary (2001). *Lexical Functional Grammar*, vol. 34 of *Syntax and semantics*. Academic Press. ISBN 0126135347.

Downing, Paula (1977). On the creation and use of english compound nouns. *Language*, 53(4):pp. 810–842.

Dyvik, Helge (1998). Translations as Semantic Mirrors. In *Proceedings of the ECAI'98 Workshop on multilinguality in the lexicon*, pp. 24–44. Brighton, UK.

Dyvik, Helge, Paul Meurer, and Victoria Rosén (2005). LFG, Minimal Recursion Semantics and translation. In *Proceedings of the LFG 2005 Conference*. To appear.

Eek, Øystein et al. (eds.) (2001). *Engelsk stor ordbok: engelsk-norsk/norsk-engelsk*. Kunnskapsforlaget, Norway. ISBN 82-573-1288-6.

Faarlund, J. T. (1977). Embedded clause reduction and Scandinavian gender agreement. *Journal of Linguistics*, 13:pp. 239–257.

Fielding, Roy Thomas (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Ph.D. thesis, University of California, Irvine.

Firth, John R. (1957). A Synopsis of Linguistic Theory 1930-1955. *Studies in Linguistic Analysis*.

Grefenstette, Gregory (1999). The WWW as a resource for example-based machine translation tasks. In *Proceedings of the ASLIB Conference on Translating and the Computer*. London.

Gudgin, Martin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Nila Mitra, and Henrik Frystyk Nielsen (2003). *SOAP Specifications*. Tech. rep., W3C.

Hasselgård, Hilde, Stig Johansson, and Per Lysvåg (1998). *English Grammar: Theory and Use*. Universitetsforlaget, Oslo, Norway. ISBN 82-00-126935.

Hovdenak, Marit (ed.) (2001). *Nynorskordboka*. Det Norske Samlaget, Oslo, Norway, 3rd edn. ISBN 8252151809.

Huddleston, Rodney D. and Geoffrey K. Pullum (eds.) (2002). *The Cambridge grammar of the English language*. Cambridge University Press. ISBN 0-521-43146-8.

Johannesen, Janne Bondi and Helge Hauglin (1998). An Automatic analysis of compounds. In Haukioja, T. (ed.), *Papers from the 16th Scandinavian Conference of Linguistics*, pp. 209–220. Turku/Åbo, Finland.

Johansen, Arnt Richard (2006). *A comparison of human and automatic evaluation of machine translation.* Master's thesis, NTNU, Trondheim, Norway. To appear.

Jurafsky, Daniel and James H. Martin (2000). *Speech and language processing.* Prentice-Hall, Inc. ISBN 0130950696.

Keller, Frank and Mirella Lapata (2003). Using the Web to Obtain Frequencies for Unseen Bigrams. *Computational Linguistics*, 29(3):pp. 459–484.

Kelsey, Richard, William Clinger, and Jonathan Rees (1998). The revised(5) report on the algorithmic language scheme. *Higher-Order and Symbolic Computation*, 11(1).

Kiselyov, Oleg (2002). A Better XML Parser through Functional Programming. In Krishnamurthi, S. and C. R. Ramakrishnan (eds.), *Practical Aspects of Declarative Languages: 4th International Symposium*, Lecture Notes in Computer Science. Springer-Verlag Heidelberg, Portland, OR. USA.

Lønning, Jan Tore, Stephan Oepen, Dorothee Beermann, Lars Hellan, John Carroll, Helge Dyvik, Dan Flickinger, Janne Bondi Johannsen, Paul Meurer, Torbjørn Nordgård, Victoria Rosén, and Erik Velldal (2004). LOGON. A Norwegian MT effort. In *Proceedings of the Workshop in Recent Advances in Scandinavian Machine Translation*, p. 6. Uppsala, Sweden.

Mayer, Tim (2005). Our blog is growing up - and so has our index. Yahoo! Search blog, `http://www.ysearchblog.com/archives/000172.html`.

Melamed, I. Dan (1997). Automatic Discovery of Non-Compositional Compounds in Parallel Data. In *Proceedings of the 2nd Conference on Empirical Methods in Natural Language Processing (EMNLP'97).* Arxiv.

Moon, Rosamund (1998). *Fixed Expressions and Idioms in English: A Corpus-Based Approach.* Clarendon Press, Oxford, UK. ISBN 0-19-823614-X.

Munthe, Synneve Kjuus (1972). *Sammensatte ord. En kvantitativ undersøkelse av norsk litteratur og sakprosa.* Master's thesis, University of Oslo/Bergen. Not available to the public.

Nordgård, Torbjørn (1996). NorKompLeks: Some Linguistic Specifications and Applications. In Lindebjerg, Ore, and Reigem (eds.), *ALLC-ACH '96. Abstracts.* Universitetet i Bergen, Humanistisk Datasenter, Bergen.

Oepen, Stephan, Helge Dyvik, Jan Tore Lønning, Erik Velldal, Dorothee Beermann, John Carroll, Dan Flickinger, Lars Hellan, Janne Bondi Johannessen, Paul Meurer, Torbjørn Nordgård, and Victoria Rosén (2004). Som å kapp-ete med trollet? Towards MRS-based Norwegian – English Machine Translation. In *Proceedings of the 10th International Conference on Theoretical and Methodological Issues in Machine Translation.* Baltimore, MD.

Papineni, K., S. Roukos, T. Ward, and W.-J. Zhu (2002). BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for the Computational Linguistics (ACL)*, pp. 311–318. Philadelphia, USA.

Pollard, Carl and Ivan A. Sag (1994). *Head-Driven Phrase Structure Grammar.* University of Chicago Press, Chicago, IL, USA.

Pullum, Geoffrey K. (2004). Snowclones: lexicographical dating to the second. Language Log, `http://itre.cis.upenn.edu/~myl/languagelog/archives/000350.html`.

Rackow, Ulrike, Ido Dagan, and Ulrike Schwall (1992). Automatic translation of noun compounds. In *Proceedings of COLING 92*, pp. 1249–1253. Nantes, France.

Ryder, Mary Ellen (1994). *Ordered Chaos: The Interpretation of English Noun-Noun Compounds.* University of California Press, Berkeley. ISBN 0520097777.

Sag, Ivan A., Timothy Baldwin, Francis Bond, Ann Copestake, and Dan Flickinger (2002). Multiword expressions: a pain in the neck for NLP. In *Proceedings of the Third International Conference on Intelligent Text Processing and Computational Linguistics (CICLING 2002)*, pp. 1–15. Mexico City, Mexico.

Searls, Doc and David Weinberger (2003). The world of ends. Website, `http://worldofends.com/`.

Søgaard, Anders (2004). A compound matrix. In Müller, Stefan (ed.), *Proceedings of the HPSG04 Conference*. CSLI Publications.

Sullivan, Danny (2005). End Of Size Wars? Google Says Most Comprehensive But Drops Home Page Count. Search Engine Watch, `http://searchenginewatch.com/searchday/article.php/3551586`.

Tanaka, Takaaki and Timothy Baldwin (2003a). Noun-noun compound machine translation: A feasibility study on shallow processing. In *Proceedings of the ACL-2003 Workshop on Multiword Expressions*, pp. 17–24.

— (2003b). Translation selection for Japanese-English Noun-Noun Compounds. In *Proceedings of Machine Translation Summit IX*, pp. 378–385. New Orleans, USA.

Tropashko, Vadim (2005). Nested intervals tree encoding in SQL. *SIGMOD Record*, 34(2).

Véronis, Jean (2005a). Web: Googlean logic. Technologies du langage, `http://aixtal.blogspot.com/2005/01/web-googlean-logic-en.html`.

— (2005b). Yahoo: Missing pages? (1). Technologies du langage, `http://aixtal.blogspot.com/2005/08/yahoo-missing-pages-1.html`.

— (2005c). Yahoo: Missing pages? (2). Technologies du langage, `http://aixtal.blogspot.com/2005/08/yahoo-missing-pages-2.html`.

— (2005d). Yahoo: Missing pages? (3). Technologies du langage, `http://aixtal.blogspot.com/2005/08/yahoo-missing-pages-3.html`.

— (2005e). Yahoo: Missing pages? (4). Technologies du langage, `http://aixtal.blogspot.com/2005/08/yahoo-missing-pages-4.html`.

Wangensteen, Boye (ed.) (2005). *Bokmålsordboka*. Kunnskapsforlaget, Norway. ISBN 8200217639.

Zgusta, L. (1971). *Manual of lexicography*. The Hague, Paris. Definition of multiword lexical unit.

BIBLIOGRAPHY

Homepages

Project	Homepage	Accessed
BNC	http://www.natcorp.ox.ac.uk/	2005-11-29
Google APIs	http://www.google.com/apis/	2005-12-06
Google	http://www.google.com/	2005-12-06
LKB	http://www.delph-in.net/lkb/	2005-01-09
LOGON	http://www.emmtee.net/	2005-11-18
LinGO ERG	http://lingo.stanford.edu/	2005-01-09
MWE-project	http://mwe.stanford.edu/	2005-11-30
NorGram	http://www.ling.uib.no/~victoria/NorGram/	2005-01-09
Norsk ordbank, interface	http://www.edd.uio.no/perl/search/search.cgi?appid=72;tabid=1106	2006-01-09
ParGram	http://www2.parc.com/istl/groups/nltt/pargram/	2005-01-09
Penn Treebank Project	http://www.cis.upenn.edu/~treebank/	2006-01-13
Redwoods	http://redwoods.stanford.edu/	2005-01-06
TIGER	http://www.ims.uni-stuttgart.de/projekte/TIGER/	2005-01-06
Tourist-corpus	http://www.hf.uio.no/tekstlab/prosjekter/tourist/	2006-01-08
Tourist-corpus, interface	http://omilia.uio.no/touristcorpus/	2006-01-08
TREPIL	http://gandalf.aksis.uib.no/trepil/index_eng.page	2006-01-09

Continued on next page

HOMEPAGES

Project	Homepage	Accessed
XLE	http://www2.parc.com/istl/groups/nltt/xle/	2005-01-09
XLE-MRS	http://decentius.hit.uib.no/logon/xle-mrs.xml	2006-01-23
Yahoo APIs	http://developer.yahoo.net/search/index.html	2005-12-06
Yahoo! Search	http://search.yahoo.com/	2005-12-06

Starts on page 127

Index

Authors

Bar-Hillel (1960), 1, 119
Bauer (2004), 38, 119
Benford (1997), 103, 119
Bergenholtz et al. (1997), 38, 119
Borthen (2003), 7, 119
Bresnan (2001), 6, 119
Butt et al. (1999), 6, 119
Cao and Li (2002), 52, 119
Carroll and Oepen (2005), 6, 8, 119
Celko (2004), 16, 120
Christensen et al. (2001), 78, 120

♢-words

Example words