GroupDetector

Jimmy Andersson
Lunds universitet
jimmy77@telia.com

Tommy Karlsson
Lunds universitet
tommy.karlsson.350@student.lu.se

1 Inledning

Som projektuppgift valde vi att göra ett program för att leta upp verb, substantiv, adjektiv och prepositionsgrupper i texter.
Mjukvaran som utvecklats i vårt projekt ska användas av CarSim-projektet som är ett system för att automatiskt konvertera skrivna beskrivningar av trafikolyckor till 3D-animeringar. CarSim projektet utvecklas av LUCAS (Center for Applied Software Research vid Lunds Universitet).
Andra användningsområden är grammatikkontroll, översättarstöd och informationsextrahering.

2 Taggar

2.1 Granska-taggar

I en text taggad enligt granska-formatet åtföljs varje ord av en tagg som beskriver vilken satsdel ordet tillhör. Exempel 2.1 visar en mening som är taggad enligt granska-formatet. För en förklaring av taggarnas betydelse, se bilaga 1.

```
Hans <ps.utr/neu.sin/plu.def> tidigare <jj.kom.utr/neu.sin/plu.ind/def.nom> grafik
<nn.utr.sin.ind.nom> har <vb.prs.akt> varit <vb.sup.akt> noggrant <ab.pos>
genomtänkt <pc.prf.utr.sin.ind.nom> och <kn> konstruerad <pc.prf.utr.sin.ind.nom>
lika <ab> målmedvetet <ab.pos> som <kn> den <dt.utr.sin.def> arkitektur
<nn.utr.sin.ind.nom> den <pn.utr.sin.def.sub/obj> skildrar <vb.prs.akt> .
```

Exempel 2.1

2.2 SUC1A-taggar

SUC1A-formatet har en något annorlunda utseende, jämför med granska-formatet. För varje ord finns information om satsdel, ordets grundform och ordningsnummer. Exempel 2.2 visar en mening som är taggad enligt SUC1A-formatet. För en förklaring av taggarnas betydelse, se bilaga 1.

```
("<Någonting>"        <140>
        (PN NEU SIN IND SUB/OBJ "någonting"))
("<har>"        <141>
        (VB PRS AKT "ha"))
("<hänt>"       <142>
        (VB SUP AKT "hända"))
("<med>"        <143>
        (PP "med"))
```

```
("<hans>"       <144>
        (PS UTR/NEU SIN/PLU DEF "hans"))
("<syn>"        <145>
        (NN UTR SIN IND NOM "syn"))
("<på>"         <146>
        (PP "på"))
("<staden>"     <147>
        (NN UTR SIN DEF NOM "stad"))
("<som>"        <148>
        (HP - - - "som"))
("<också>"      <149>
        (AB "också"))
("<fått>"       <150>
        (VB SUP AKT "få"))
("<följder>"    <151>
        (NN UTR PLU IND NOM "följd"))
("<för>"        <152>
        (PP "för"))
("<hans>"       <153>
        (PS UTR/NEU SIN/PLU DEF "hans"))
("<grafik>"     <154>
        (NN UTR SIN IND NOM "grafik"))
("<.>"  <155>
        (DL MAD "."))
```

Exempel 2.2

2.3 Våra taggar

Våra taggar beskriver olika grupper av ord i texten. Början på en grupp markeras med en tagg på formen <XX> och avslutas med </XX>. Exempel 2.3 visar en mening som är taggad enligt vårt format.

Taggarna är följande:

- AG: adjektivgrupp
- PG: prepositionsgrupp
- NG: substantivgrupp
- VG: verbgrupp

```
<AG> Omkullvält </AG> <PG> i Östeuropa </PG> och illa ute <PG> i
Sovjetunionen </PG> <VG> dyker </VG> <NG> den f_d kuppmakaren </NG> och
<NG> frälsargestalten </NG> upp <PG> i 117 olika skepnader </PG> <VG> där
</VG> <NG> granskogen </NG> <VG> susar </VG> och <NG> sjön </NG> <VG>
ligger </VG> <AG> blank </AG> inte långt <PG> från Jönköping </PG> .
```

Exempel 2.3

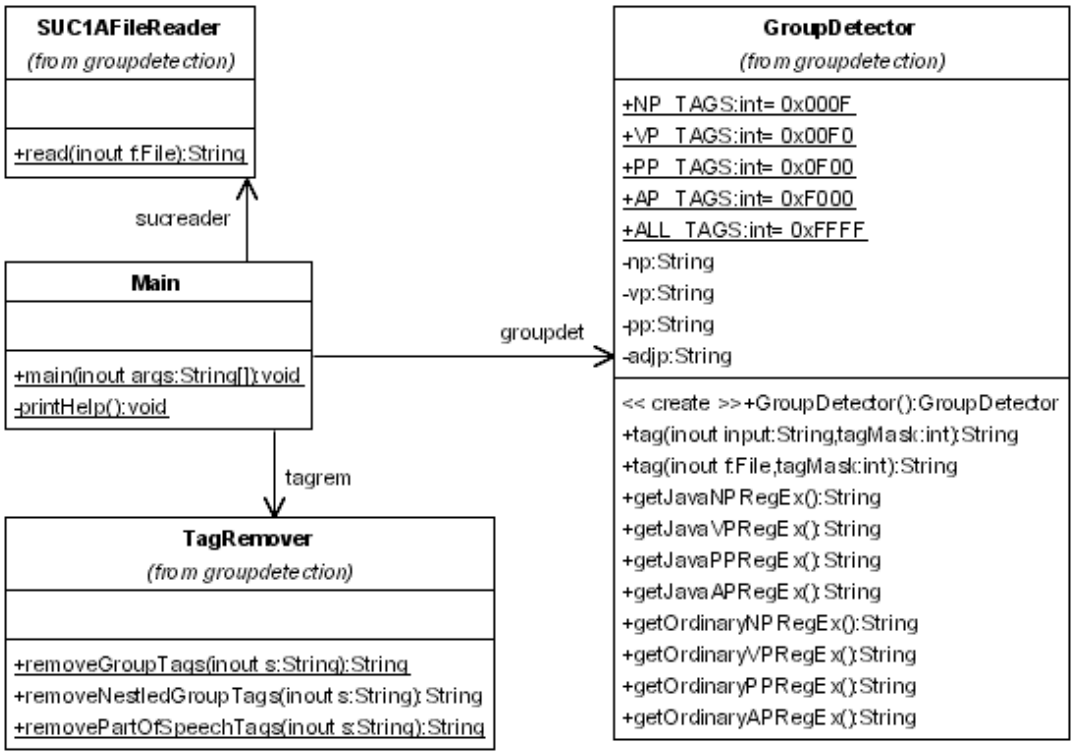
3 Systemet

3.1 Struktur

Programmet är uppdelat i 3 olika klasser, *GroupDetector*, *TagRemover* och *SUC1AFileReader*.

- Klassen GroupDetector används för att leta upp ordgrupper i en text.
- Klassen TagRemover används för att ta bort taggar i en text så att texten blir lättare att läsa.
- Klassen SUC1AFileReader används för att läsa in filer med SUC1A-formatet och formatera om det till granska-formatet.

Main-klassen används för att testa de olika systemen och de olika klasserna.



Figur 3.1 – Klassdiagram

3.2 Reguljära uttryck

- NP = [[([DET]+) [PRON]+] | [DET ADJ] | [([DET]+) (ADVPOS) (NUM) (ADJP) (NUM) [NOUN]+] | [(DET) POSS_PRON (ADJP) [NOUN]+] ;
- VP = [[INT_REL_ADV] | [(INF) [VERB]+ (PART)] | [AUX NP [VERB]+ (PART)]] ;
- PP = [[[PREP] | [PREP KONJ PREP]] NP] ;
- AP = [([ADV]+) [ADJ]+] ;

4 Utvärdering

För att utvärdera projektet har vi beräknat värden för precision och recall enligt följande formler;

precision = antal korrekta / antal försök
recall = antal korrekta / totalt antal

Vi fick följande resultat:
Precision: 0.949152542372881
Recall: 0.96551724137931

Texten som vi använde vid mätningen finns i bilaga 3.
Vi är väldigt nöjda med resultatet. Det blev lite bättre än vi hade förväntat oss men man ska nog testa med större texter innan man kan dra alltför stora slutsatser.

5 Referenser

Steven Abney, *Chunk Stylebook*, 1996
http://www.vinartus.net/spa/96i.pdf

Victoria Johansson, *NP-detektion*, 2000
http://www.nada.kth.se/theory/projects/granska/rapporter/vicuppsats.pdf

Beáta Megyesi & Sara Rydin, *Towards a Finite-State Parser for Swedish*,
http://www.speech.kth.se/%7Ebea/final-megyesi-rydin.pdf

6 Bilaga 1 - Taggar i Granska & SUC

Category Code	Category
AB	Adverb
DL	Delimiter (Punctuation)
DT	Determiner
HA	Interrogative/Relative Adverb
HD	Interrogative/Relative Determiner
HP	Interrogative/Relative Pronoun
HS	Interrogative/Relative Possessive
IE	Infinitive Marker
IN	Interjection
JJ	Adjective
KN	Conjunction
NN	Noun
PC	Participle
PL	Particle
PM	Proper Noun
PN	Pronoun
PP	Preposition
PS	Possessive
RG	Cardinal Number
RO	Ordinal Number
SN	Subjunction
UO	Foreign Word
VB	Verb

Feature Code	Feature	
UTR	Common (Utrum)	Gender
NEU	Neutre	Gender
MAS	Masculine	Gender
UTR/NEU	Underspecified	Gender
-	Unspecified	Gender
SIN	Singular	Number
PLU	Plural	Number
SIN/PLU	Underspecified	Number
-	Unspecified	Number
IND	Indefinite	Definiteness
DEF	Definite	Definiteness
IND/DEF	Underspecified	Definiteness
-	Unspecified	Definiteness
NOM	Nominative	Case
GEN	Genitive	Case
SMS	Compound	Case
-	Unspecified	Case
POS	Positive	Degree
KOM	Comparative	Degree
SUV	Superlative	Degree
SUB	Subject	Pronoun Form
OBJ	Object	Pronoun Form
SUB/OBJ	Underspecified	Pronoun Form
PRS	Present	Verb Form
PRT	Preterite	Verb Form
INF	Infinitive	Verb Form
SUP	Supinum	Verb Form
IMP	Imperative	Verb Form
AKT	Active	Voice
SFO	S-form	Voice
KON	Subjunctive	Mood
PRF	Perfect	Perfect
AN	Abbreviation	Form

7 Bilaga 2 – Reguljära uttryck

NP

```
((((([\S]+) <dt[^>]*>\s+)?((([\S]+) <pn[^>]*>\s+)+))|((([\S]+) <dt[^>]*>\s+)(([\S]+)
<((jj)|(pc\.prf))[^>]*>\s+))|((([\S]+) <dt[^>]*>\s+)?(([\S]+)
<ab.pos[^>]*>\s+)?((([\S]+) <r[^>]*>\s+)+)?((((([\S]+) <ab[^>]*>\s+)+)?((([\S]+)
<((jj)|(pc\.prf))[^>]*>\s+)+))?((([\S]+) <r[^>]*>\s+)+)?((([\S]+)
<((nn)|(pm))[^>]*>\s+)+))|((([\S]+) <dt[^>]*>\s+)?(([\S]+) <ps[^>]*>\s+)((((([\S]+)
<ab[^>]*>\s+)+)?((([\S]+) <((jj)|(pc\.prf))[^>]*>\s+)+))?((([\S]+)
<((nn)|(pm))[^>]*>\s+)+)))+
```

VP

```
((([\S]+) <ha[^>]*>\s+))|((([\S]+) <ie[^>]*>\s+)?((([\S]+) <vb[^>]*>\s+)+)(([\S]+)
<pc[^>]*>\s+)?)|((([\S]+) <vb[^>]*>\s+)((((([\S]+) <dt[^>]*>\s+)?((([\S]+)
<pn[^>]*>\s+)+))|((([\S]+) <dt[^>]*>\s+)(([\S]+) <((jj)|(pc\.prf))[^>]*>\s+))|((([\S]+)
<dt[^>]*>\s+)?(([\S]+) <ab.pos[^>]*>\s+)?((([\S]+) <r[^>]*>\s+)+)?((((([\S]+)
<ab[^>]*>\s+)+)?((([\S]+) <((jj)|(pc\.prf))[^>]*>\s+)+))?((([\S]+)
<r[^>]*>\s+)+)?((([\S]+) <((nn)|(pm))[^>]*>\s+)+))|((((([\S]+) <dt[^>]*>\s+)?(([\S]+)
<ps[^>]*>\s+)((((([\S]+) <ab[^>]*>\s+)+)?((([\S]+)
<((jj)|(pc\.prf))[^>]*>\s+)+))?((((([\S]+) <((nn)|(pm))[^>]*>\s+)+)))+((([\S]+)
<vb[^>]*>\s+)+)(([\S]+) <pc[^>]*>\s+)?)
```

AP

```
((((([\S]+) <ab[^>]*>\s+)+)?((([\S]+) <((jj)|(pc\.prf))[^>]*>\s+)+))
```

PP

```
((((([\S]+) <pp[^>]*>\s+))|((([\S]+) <pp[^>]*>\s+)(([\S]+) <kn[^>]*>\s+)(([\S]+)
<pp[^>]*>\s+)))((((([\S]+) <dt[^>]*>\s+)?((([\S]+) <pn[^>]*>\s+)+))|((([\S]+)
<dt[^>]*>\s+)(([\S]+) <((jj)|(pc\.prf))[^>]*>\s+))|((((([\S]+) <dt[^>]*>\s+)?(([\S]+)
<ab.pos[^>]*>\s+)?((([\S]+) <r[^>]*>\s+)+)?((((([\S]+) <ab[^>]*>\s+)+)?((([\S]+)
<((jj)|(pc\.prf))[^>]*>\s+)+))?((([\S]+) <r[^>]*>\s+)+)?((([\S]+)
<((nn)|(pm))[^>]*>\s+)+))|((((([\S]+) <dt[^>]*>\s+)?(([\S]+) <ps[^>]*>\s+)((((([\S]+)
<ab[^>]*>\s+)+)?((([\S]+) <((jj)|(pc\.prf))[^>]*>\s+)+))?((([\S]+)
<((nn)|(pm))[^>]*>\s+)+)))+
```

8 Bilaga 3 – Test-text

Skillnaden mellan vårt resultat och facit har markerats med fet stil.

Facit

<NG> Syfte </NG> och <NG> fr}gest{llningar </NG> . Mer {n <NG> ett kvarts sekel </NG> <VG> har passerat </VG> sedan <NG> de f|rsta jugoslaverna </NG> <VG> slog </VG> <NG> sig </NG> ner <PG> i Stockholm </PG> (1) . D} , <PG> vid mitten </PG> <PG> av 60-talet </PG> , <VG> s}gs </VG> <NG> arbetskraftsinvandringen </NG> mest som <NG> en tillf{llig l|sning </NG> <PG> p} tillf{lliga problem </PG> ; efter <VG> att ha tj{nat </VG> ihop tillr{ckligt <VG> skulle </VG> <NG> de flesta </NG> <VG> flytta </VG> hem igen . S} <VG> var </VG> <NG> det </NG> <VG> t{nkt </VG> , b}de **<PG> fr}n svenskt </PG>** och <NG> jugoslaviskt myndighetsh}ll </NG> . Men <NG> verkligheten </NG> <VG> blev </VG> **<AG> en annan </AG>** . Bara <NG> n}gra f} </NG> <VG> v{nde </VG> hem igen . [ven om <NG> m}nga </NG> {nnu <VG> har </VG> <NG> starka band </NG> <PG> till hemlandet </PG> och <NG> n}gra </NG> {nnu <VG> n{r </VG> <NG> dr|mmen </NG> <PG> om }terv{ndandet </PG> , s} <VG> har </VG> <NG> de flesta </NG> {nd} <VG> valt </VG> <VG> att bos{tta </VG> <NG> sig </NG> <PG> i Stockholm </PG> **f|r gott** . <PG> Vid slutet </PG> <PG> av 1980-talet </PG> <VG> bodde </VG> <NG> drygt 8000 jugoslaviska medborgare </NG> <PG> i Stockholmstrakten </PG> . <NG> En stor del serber </NG> , men d{rut|ver ocks} <NG> kroater </NG> , <NG> bosnier </NG> , <NG> slovener </NG> , <NG> makedonier </NG> , <NG> montenegriner </NG> , <NG> ungrare </NG> , <NG> vlacher </NG> , <NG> albaner </NG> , <NG> rusiner </NG> och <NG> slovaker </NG>

Vårt resultat

<NG> Syfte </NG> och <NG> fr}gest{llningar </NG> . Mer {n <NG> ett kvarts sekel </NG> <VG> har passerat </VG> sedan <NG> de f|rsta jugoslaverna </NG> <VG> slog </VG> <NG> sig </NG> ner <PG> i Stockholm </PG> (1) . D} , <PG> vid mitten </PG> <PG> av 60-talet </PG> , <VG> s}gs </VG> <NG> arbetskraftsinvandringen </NG> mest som <NG> en tillf{llig l|sning </NG> <PG> p} tillf{lliga problem </PG> ; efter <VG> att ha tj{nat </VG> ihop tillr{ckligt <VG> skulle </VG> <NG> de flesta </NG> <VG> flytta </VG> hem igen . S} <VG> var </VG> <NG> det </NG> <VG> t{nkt </VG> , b}de **fr}n <AG> svenskt </AG>** och <NG> jugoslaviskt myndighetsh}ll </NG> . Men <NG> verkligheten </NG> <VG> blev </VG> **<NG> en annan </NG>** . Bara <NG> n}gra f} </NG> <VG> v{nde </VG> hem igen . [ven om <NG> m}nga </NG> {nnu <VG> har </VG> <NG> starka band </NG> <PG> till hemlandet </PG> och <NG> n}gra </NG> {nnu <VG> n{r </VG> <NG> dr|mmen </NG> <PG> om }terv{ndandet </PG> , s} <VG> har </VG> <NG> de flesta </NG> {nd} <VG> valt </VG> <VG> att bos{tta </VG> <NG> sig </NG> <PG> i Stockholm </PG> **f|r <AG> gott </AG>** . <PG> Vid slutet </PG> <PG> av 1980-talet </PG> <VG> bodde </VG> <NG> drygt 8000 jugoslaviska medborgare </NG> <PG> i Stockholmstrakten </PG> . <NG> En stor del serber </NG> , men d{rut|ver ocks} <NG> kroater </NG> , <NG> bosnier </NG> , <NG> slovener </NG> , <NG> makedonier </NG> , <NG> montenegriner </NG> , <NG> ungrare </NG> , <NG> vlacher </NG> , <NG> albaner </NG> , <NG> rusiner </NG> och <NG> slovaker </NG>

9 Bilaga 4 - Källkod

Här följer källkoden i projektet.

GroupDetector

```
package groupdetection;
import java.io.*;
import java.util.regex.*;
import java.util.*;

/**
 * <p>Title: Noun and Verb group detector</p>
 * <p>Description: This class supplies methods for adding NG and VG tags to
a part-of-speech tagged text.</p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: Lund University</p>
 * @author Tommy Karlsson & Jimmy Andersson
 * @version 1.0
 */

public class GroupDetector {
  public static final int NP_TAGS  = 0x000F;
  public static final int VP_TAGS  = 0x00F0;
  public static final int PP_TAGS  = 0x0F00;
  public static final int AP_TAGS  = 0xF000;
  public static final int ALL_TAGS = 0xFFFF;

  private String np;   // beskriver det regexp som används för att matcha
noun-phrase
  private String vp;   // beskriver det regexp som används för att matcha
verb-phrase
  private String pp;   // beskriver det regexp som används för att matcha
preposition-phrase
  private String adjp; // beskriver det regexp som används för att matcha
adjective-phrase


  /**
   * Sets up the strings used to compile the regular expressions.
   */
  public GroupDetector() {

    // matchar ett ord
    String anyword="(?:[\\S]+)";
    // matchar ett adverb
    String adverb="(?:"+anyword+" <ab[^>]*>\\s+)";
    // matchar ab.pos
    String adverb_pos="(?:"+anyword+" <ab.pos[^>]*>\\s+)";
    // matchar ett interrogativit / relativt adverb
    String inter_rel_adverb="(?:"+anyword+" <ha[^>]*>\\s+)";
    // matchar ett adjektiv
    String adjective="(?:"+anyword+" <(?:(?:jj)|(?:pc\\.prf))[^>]*>\\s+)";
    // matchar en preposition
    String prep="(?:"+anyword+" <pp[^>]*>\\s+)";
    // matchar en konjunktion
    String konj="(?:"+anyword+" <kn[^>]*>\\s+)";
    // matchar en sekvens av pronomen
    String pronouns="(?:(?:"+anyword+" <pn[^>]*>\\s+)+)";
    // matchar ett possesivt pronomen
    String posspron="(?:"+anyword+" <ps[^>]*>\\s+)";
    // matchar en determinant
    String det="(?:"+anyword+" <dt[^>]*>\\s+)";
    // matchar en sekvens av räkneord
    String numerals="(?:(?:"+anyword+" <r[^>]*>\\s+)+)";
```

```
    // matchar en sekvens av substantiv
    String nouns="(?:(?:"+anyword+" <(?:(?:nn)|(?:pm))[^>]*>\\s+)+)";
    // matchar en infinitiv-markör
    String inf="(?:"+anyword+" <ie[^>]*>\\s+)";
    // matchar en sekvens av verb
    String verbs="(?:(?:"+anyword+" <vb[^>]*>\\s+)+)";
    // matchar participle
    String participle="(?:"+anyword+" <pc[^>]*>\\s+)";
    // matchar aux
    String aux="(?:"+anyword+" <vb[^>]*>\\s+)";
    // matchar en adj-phrase
    adjp="(?:(?:"+adverb+"+)?(?:"+adjective+"+))";
    // matchar en noun-phrase
    np =
"(?:(?:"+det+"?"+pronouns+")|(?:"+det+adjective+")|(?:"+det+"?"+adverb_pos+"
?"+numerals+"?"+adjp+"?"+numerals+"?"+nouns+")|(?:"+det+"?"+posspron+adjp+"?
"+nouns+"))+";
    // matchar en verb-phrase

vp="(?:"+inter_rel_adverb+")|(?:"+inf+"?"+verbs+participle+"?)|(?:"+aux+np+v
erbs+participle+"?)";
    // matchar en prep-phrase
    pp="(?:(?:"+prep+")|(?:"+prep+konj+prep+"))"+np;

//    define NP [ [([DET]+) [PRON]+] | [DET ADJ] | [([DET]+) (ADVPOS) (NUM)
(ADJP) (NUM) [NOUN]+] | [(DET) POSS_PRON (ADJP) [NOUN]+] ;
//    define VP [ [INT_REL_ADV] | [(INF) [VERB]+ (PART)] | [AUX NP [VERB]+
(PART)]] ;
//    define PP [ [[PREP] | [PREP KONJ PREP]] NP] ;
//    define AP [ ([ADV]+) [ADJ]+] ;
  }

  /**
   * Detects noun- and verb-groups in a text, and add appropriate tags.
   * All original tags and text are left untouched.
   *
   * @param s The text to be tagged.
   * @param tagMask The mask of the tags to be added.
   * @return The text with the specified tags.
   */
  public String tag(String input, int tagMask) {
    String taggedString="";
    Vector tag_vector=new Vector();

    try {
      Pattern p;
      Matcher m;
      if((tagMask & NP_TAGS)==NP_TAGS) {
        // kompilera np-regex och skapa en matcher
        p = Pattern.compile(np);
        m = p.matcher(input);
        // applicera np-regex på strängen och spara alla taggar på np-
stacken
        while (m.find()) {
          tag_vector.add(new GroupTag(GroupTag.NG_BEGIN, m.start()));
          tag_vector.add(new GroupTag(GroupTag.NG_END, m.end()));
        }
      }
      if((tagMask & VP_TAGS)==VP_TAGS) {
        // kompilera vp-regex och skapa en matcher
        p = Pattern.compile(vp);
        m = p.matcher(input);
        // applicera vp-regex på strängen och spara alla taggar på vp-
stacken
        while (m.find()) {
          tag_vector.add(new GroupTag(GroupTag.VG_BEGIN, m.start()));
```

```
          tag_vector.add(new GroupTag(GroupTag.VG_END, m.end()));
        }
      }
      if((tagMask & PP_TAGS)==PP_TAGS) {
        // kompilera pp-regex och skapa en matcher
        p = Pattern.compile(pp);
        m = p.matcher(input);
        // applicera pp-regex på strängen och spara alla taggar på pp-
stacken
        while (m.find()) {
          tag_vector.add(new GroupTag(GroupTag.PP_BEGIN, m.start()));
          tag_vector.add(new GroupTag(GroupTag.PP_END, m.end()));
        }
      }
      if((tagMask & AP_TAGS)==AP_TAGS) {
        // kompilera adjp-regex och skapa en matcher
        p = Pattern.compile(adjp);
        m = p.matcher(input);
        // applicera adjp-regex på strängen och spara alla taggar på pp-
stacken
        while (m.find()) {
          tag_vector.add(new GroupTag(GroupTag.ADJP_BEGIN, m.start()));
          tag_vector.add(new GroupTag(GroupTag.ADJP_END, m.end()));
        }
      }

      // joxa lite för att sortera taggarna
      tag_vector.trimToSize();
      Object[] tagv = tag_vector.toArray();
      Arrays.sort(tagv);
      List tag_list=Arrays.asList(tagv);

      // lägg till taggarna i strängen, börja bakifrån och arbeta mot
strängens början.
      GroupTag g;
      for(int i=tag_list.size()-1;i>=0;i--) {
        g=(GroupTag)tag_list.get(i);
        input=input.substring(0,g.getPos()) + g +
input.substring(g.getPos());
      }
      taggedString = input;
    }
    catch(Exception e) {System.out.println(e.getMessage());
e.printStackTrace();}
    return taggedString;
  }

  /**
   * Adds support for tagging text stored in a file.
   * The idea is obviously to capture the file-reading inside this class.
   * Any user of this method is responsible for supplying a file with
correctly formatted text.
   * Note that the actual tagging is performed by a call to {@link
GroupDetector.tag(String) tag(String)}, hence
   * this method should deliver exactly the same result as if you do the
file-reading yourself.
   * Also note that this method has a limitation on file-size: files may not
be any larger than 2^31-1 bytes.
   *
   * @param f The file with the text to be tagged.
   * @param tagMask The mask of the tags to be added.
   * @return The text with the specified tags.
   *
   */
  public String tag(File f, int tagMask) {
    // check if file is readable
```

```
    if(!f.canRead()) {
      System.out.println("Can't read file: "+f.getAbsolutePath());
      return "";
    }
    try {
      // read the file
      FileReader fr=new FileReader(f);
      char[] cv=new char[(int)f.length()];
      fr.read(cv);
      fr.close();
      String s=new String(cv);
      // do the tagging
      return tag(s, tagMask);
    }
    catch(Exception e) { System.out.println(e.getMessage()); return "";}
  }

  /**
   * Get the java-formatted string describing the regexp used to match noun-
phrases.
   * @return The actual string used to compile the regexp.
   */
  public String getJavaNPRegEx() {
    return np;
  }

  /**
   * Get the java-formatted string describing the regexp used to match verb-
phrases.
   * @return The actual string used to compile the regexp.
   */
  public String getJavaVPRegEx() {
    return vp;
  }

  /**
   * Get the java-formatted string describing the regexp used to match
pronoun-phrases.
   * @return The actual string used to compile the regexp.
   */
  public String getJavaPPRegEx() {
    return pp;
  }

  /**
   * Get the java-formatted string describing the regexp used to match
adjective-phrases.
   * @return The actual string used to compile the regexp.
   */
  public String getJavaAPRegEx() {
    return adjp;
  }

  /**
   * Get a perl-like version of the regexp used to match noun-phrases.
   * Note that the escape character is escaped.
   * @return A perl-like-formatted version of the regexp.
   */
  public String getOrdinaryNPRegEx() {
    return np.replaceAll("\\?:","");
  }

  /**
   * Get a perl-like version of the regexp used to match verb-phrases.
   * Note that the escape character is escaped.
   * @return A perl-like-formatted version of the regexp.
```

```
   */
  public String getOrdinaryVPRegEx() {
    return vp.replaceAll("\\?:","");
  }

  /**
   * Get a perl-like version of the regexp used to match pronoun-phrases.
   * Note that the escape character is escaped.
   * @return A perl-like-formatted version of the regexp.
   */
  public String getOrdinaryPPRegEx() {
    return pp.replaceAll("\\?:","");
  }

  /**
   * Get a perl-like version of the regexp used to match adjective-phrases.
   * Note that the escape character is escaped.
   * @return A perl-like-formatted version of the regexp.
   */
  public String getOrdinaryAPRegEx() {
    return adjp.replaceAll("\\?:","");
  }

  /**
   * Internal class used to represent a group-tag.
   */
  private class GroupTag implements Comparable {
    // tänk på att taggarnas nummer påverkar deras ordning när de sorteras!
    public static final int VG_BEGIN=1;
    public static final int ADJP_BEGIN=2;
    public static final int NG_BEGIN=3;
    public static final int PP_BEGIN=4;
    public static final int VG_END=5;
    public static final int ADJP_END=6;
    public static final int NG_END=7;
    public static final int PP_END=8;

    private int type;
    private int pos;
    public GroupTag(int t, int p) {
      type=t;
      pos=p;
    }
    public int compareTo(Object _tag) {
      GroupTag tag=(GroupTag)_tag;
      if(pos<tag.getPos()) { return -1; }
      if(pos>tag.getPos()) { return 1; }
      else {
        if(this.isOpening() && tag.isClosing()) { return 1; }
        else if(this.isClosing() && tag.isOpening()) { return -1; }
        else if(this.isOpening() && tag.isOpening() && this.type <
tag.getType()) { return  1; }
        else if(this.isOpening() && tag.isOpening() && this.type >
tag.getType()) { return -1; }
        else if(this.isClosing() && tag.isClosing() && this.type <
tag.getType()) { return -1; }
        else if(this.isClosing() && tag.isClosing() && this.type >
tag.getType()) { return  1; }
        else return 0;
      }
    }
    public int getPos() {return pos;}
    private int getType() {return type;}
    public boolean isOpening() { return (type==1 || type==2 || type==3 ||
type==4); }
    public boolean isClosing() { return !isOpening(); }
```

```
    public String toString() {
      switch (type) {
        case VG_BEGIN :
          return "<VG> ";
        case VG_END :
          return "</VG> ";
        case NG_BEGIN :
          return "<NG> ";
        case NG_END :
          return "</NG> ";
        case PP_BEGIN :
          return "<PG> ";
        case PP_END :
          return "</PG> ";
        case ADJP_BEGIN :
          return "<AG> ";
        case ADJP_END :
          return "</AG> ";
        default :
          return "";
      }
    }
  }
}
```

SUC1AFileReader

```
package groupdetection;
import java.io.*;
import java.util.regex.*;

/**
 * <p>Title: Noun and Verb group detector</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: Lund University</p>
 * @author Tommy Karlsson & Jimmy Andersson
 * @version 1.0
 */
public class SUC1AFileReader {
  /**
   * Reads a file on the SUC1A-format and returns the text with part-of-
speech-tags on the Granska-format.
   * All headers and other "non-sense" information is not present in the
resulting string. Also, all newlines are removed.
   *
   * @param f File-object refering to a file on the SUC1A-format.
   * @return A string with part-of-speech-tags on the Granska-format.
   */
  public static String read(File f) {
    String fs="";
    try {
      FileReader fr=new FileReader(f);
      char[] cv=new char[(int)f.length()];
      fr.read(cv);
      fr.close();
      String s=new String(cv);
      Pattern p;
      Matcher m;
      //remove all header/bogus lines
      s=s.replaceAll("(?m)^\\(\"<<.*?$[\\n\\r]*","");
      //remove all newlines
      s=s.replaceAll("(?m)[\\r\\n]+","");
      //a somewhat complex (at least hard to read) regexp, and unfortunately
not very efficient
```

```
      p=Pattern.compile("\\(\"<(.*?)>\"[^\\(]*\\(([^\"]*)\\s");
      m=p.matcher(s);
      while(m.find()) {
        fs+=m.group(1)+" ";
        fs+="<"+m.group(2).replaceAll("\\s",".").toLowerCase()+"> ";
      }
    }
    catch(Exception e) {System.out.println(e.getMessage());
e.printStackTrace();}
    return fs;
  }
}
```

TagRemover

```
package groupdetection;
import java.util.regex.*;
import java.util.*;

/**
 * <p>Title: Noun and Verb group detector</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: Lund University</p>
 * @author Tommy Karlsson & Jimmy Andersson
 * @version 1.0
 */

public class TagRemover {
  /**
   * Removes all group-tags from a string.
   *
   * @param s The string from which you want to remove the group tags.
   * @return The supplied string with all group tags removed.
   */
  public static String removeGroupTags(String s) {
    return
s.replaceAll("(?m)<((NG)|(/NG)|(VG)|(/VG)|(PG)|(/PG)|(AG)|(/AG))>\\s*","");
  }
  /**
   * Removes nestled group-tags from a string.
   * Note that this method is not guaranteed to be correct.
   *
   * @param s The string from which you want to remove the nestled group
tags.
   * @return The supplied string with all nestled group tags removed.
   */
  public String removeNestledGroupTags(String s) {
    Stack tag_stack = new Stack();
    Pattern p=Pattern.compile("<PG>[^<]*(<[/]?(?:(?:AG)|(?:NG))>)");
    Matcher m=p.matcher(s);
    while(m.find()) {
      tag_stack.push(new TagPos(m.start(1)-1,m.end(1)));
    }
    while(!tag_stack.isEmpty()) {
      TagPos tp=(TagPos)tag_stack.pop();
      s=s.substring(0,tp.getStart())+s.substring(tp.getEnd());
    }
    m=p.matcher(s);
    while(m.find()) {
      tag_stack.push(new TagPos(m.start(1)-1,m.end(1)));
    }
    while(!tag_stack.isEmpty()) {
      TagPos tp=(TagPos)tag_stack.pop();
      s=s.substring(0,tp.getStart())+s.substring(tp.getEnd());
```

```
    }
    m=p.matcher(s);
    while(m.find()) {
      tag_stack.push(new TagPos(m.start(1)-1,m.end(1)));
    }
    while(!tag_stack.isEmpty()) {
      TagPos tp=(TagPos)tag_stack.pop();
      s=s.substring(0,tp.getStart())+s.substring(tp.getEnd());
    }
    m=p.matcher(s);
    while(m.find()) {
      tag_stack.push(new TagPos(m.start(1)-1,m.end(1)));
    }
    while(!tag_stack.isEmpty()) {
      TagPos tp=(TagPos)tag_stack.pop();
      s=s.substring(0,tp.getStart())+s.substring(tp.getEnd());
    }
    p=Pattern.compile("<NG>[^<]*(<[/]?AG>)");
    m=p.matcher(s);
    while(m.find()) {
      tag_stack.push(new TagPos(m.start(1)-1,m.end(1)));
    }
    while(!tag_stack.isEmpty()) {
      TagPos tp=(TagPos)tag_stack.pop();
      s=s.substring(0,tp.getStart())+s.substring(tp.getEnd());
    }
    m=p.matcher(s);
    while(m.find()) {
      tag_stack.push(new TagPos(m.start(1)-1,m.end(1)));
    }
    while(!tag_stack.isEmpty()) {
      TagPos tp=(TagPos)tag_stack.pop();
      s=s.substring(0,tp.getStart())+s.substring(tp.getEnd());
    }
    return s;
  }

  /**
   * Removes all part-of-speech tags from a string.
   * Note that this method is based on the fact that all part-of-speech-tags
are all lowercase.
   *
   * @param s The string from which you want to remove the part-of-speech-
tags.
   * @return The supplied string with all part-of-speech-tags removed.
   */
  public static String removePartOfSpeechTags(String s) {
    return s.replaceAll("(?m)\\s<[a-z\\.\\-/&&[^ANPVG]]*?>","");
  }

  private class TagPos {
    int start;
    int end;
    public TagPos(int s, int e) {
      start=s;
      end=e;
    }
    public int getStart() {return start;}
    public int getEnd() {return end;}
  }
}
```