

STEVIN-IRME WP5, deliverable 5.1a

Report on integrating ECM Lexical database in Alpino system

Nicole Grégoire
Uil-OTS, Utrecht University
Nicole.Gregoire@let.uu.nl

November 5, 2007

Contents

1	Introduction	3
1.1	Required files	3
1.2	Approach taken	3
1.3	Alpino	4
1.4	Limitations of Alpino	5
2	Conversion procedure	8
3	Example conversions	16
3.1	EC1	16
3.2	EC2	17
3.3	EC3	19
3.4	EC4	20
3.5	EC5	21
3.6	EC6	22
	References	24
A	Conversion scripts and input information	25
A.1	template.awk	25
A.2	printPatterns.awk	29
A.3	alpinclusion.awk	35
A.4	File structure of <i>mwes.tsv</i>	38
B	Converted ECs	39
C	Excluded ECs	60

1 Introduction

This document describes the implementation of the *MWE lexicon for Dutch* in Alpino.¹ Alpino is a dependency parser for Dutch, which uses linguistic knowledge and various heuristics to construct appropriate linguistic structures of Dutch sentences.

In Alpino MWEs are stored as fixed expressions in the lexicon. The implementation of the *MWE Lexicon for Dutch* in the Alpino system comprises adding new lexical entries to the relevant sublexica. Since the Alpino grammar stays untouched, solely types of fixed expression already present in the Alpino lexicon can be implemented. This means that a good notion of the Alpino lexicon is crucial.

Although the *MWE lexicon for Dutch* contains both verbal MWEs and non-verbal MWEs, the conversion to the Alpino lexicon solely includes verbal MWEs. Limited knowledge of the treatment of non-verbal MWEs in Alpino and lack of time to acquire the required expertise are the main reasons for focusing exclusively on verbal MWEs.

The remainder of this section discusses the files required for the conversion, the approach taken, the Alpino lexicon, and the limitations of Alpino. The conversion procedure is described in section 2, whereas an illustration of the actual conversion of some classes of verbal MWEs is given in section 3. The conversion scripts are listed in appendix A. Appendix B gives an overview the classes of MWEs converted to Alpino, and appendix C lists the classes that are excluded from the conversion.

It is assumed that readers of this document have knowledge of the description fields that are part of standard representation of MWEs and MWE patterns. The standard representation is described in detail in the *Encoding Protocol* (Grégoire, 2007b).

1.1 Required files

For the conversion the following files are needed:

- from the package *MWE Lexicon for Dutch.zip*:²
 - data\mwes.tsv database with MWE descriptions
 - data\patterns.tsv database with MWE pattern descriptions
- from the package *Alpino.tar*:³
 - Lexicon\verbs.pl for expressions headed by a verb (verbal MWEs)

1.2 Approach taken

There are (at least) two approaches to implement the standard lexicon:

1. Following the spirit of the Equivalence Class Method (ECM), introduced by Odijk (2003), i.e. converting each Equivalence Class individually.⁴
 - Advantage: more control, yielding basically an error-free result

¹<http://www.let.rug.nl/vannoord/alp/Alpino/>

²Available at <http://www.tst.inl.nl/>

³Freely available at <http://www.let.rug.nl/vannoord/alp/Alpino/>

⁴In the *MWE lexicon for Dutch* expressions are grouped according to their pattern. MWEs with the same pattern form so-called Equivalence Classes (ECs) (Grégoire, 2007a).

- Disadvantages: requires more manual effort, one script for each EC conversion
2. Creating a general mapping between the standard pattern notation and the Alpino notation, and convert the instances of all Equivalence Classes at once.
 - Advantages: probably faster than approach 1, one conversion script
 - Disadvantage: error sensitive, i.e. special phenomena of ECs may be overlooked

The latter approach was taken for the conversion of a preliminary version of the lexicon. This preliminary version contained relatively simple expressions and at the point of the conversion try-out, a one-to-one mapping seemed the most appropriate way to convert the standard representation to the Alpino representation. The final representation of MWEs in the lexicon is more sophisticated and includes a range of linguistic phenomena not covered by the Alpino representation. Since the presence of some phenomena is EC dependent, the approach taken in the current conversion is the approach that follows the spirit of the ECM, viz. take one instance of an EC, define and formalize the conversion of this instance, and use the gathered information to automate the conversion of all other instances of the same EC.

1.3 Alpino

The lexicon of Alpino consists of various sublexica. Verbal MWEs must be converted to the sublexicon *verbs.pl*. *verbs.pl* contains so-called verb tags. Each verb tag specifies three types of properties:⁵

1. The tense and conjugation information (*sg1*, *sg3*, *pl*, *past(sg)*, *past(pl)*, *psp*, *inf*).
2. The auxiliary verb with which the verb conjugates (*h(ebben)*, *z(ijn)*, or *b(oth)*).
3. The possible complements.

The possible complements are represented with predefined labels that can be followed by lexemes between round brackets. For the representation of verbal MWEs the predefined label *fixed* is used, i.e. verbal MWEs are listed as fixed expressions under the verb that is the head of the expression. An example of a part of a verb tag that contains fixed complements is given in (1).

- (1) `v(draai,draait,draaien,gedraaid,draaide,draaiden,
 [z([part_intransitive(om),
 part_ld_pp(om)]),
 h([transitive,
 intransitive,
 1: part_fixed(om,[{acc(hand),pc(voor)}]),no_passive),
 2: part_fixed(aan,[acc(duimschroef),dat],norm_passive),
 3: part_fixed(aan,[acc(duimschroef)],norm_passive),
 4: fixed([acc(loer),dat],imp_passive),
 iemand een rad voor ogen draaien -->
 5: fixed([voor,ogen],[een,rad],dat],imp_passive),`

⁵<http://www.let.rug.nl/vannoord/DCOI/AnnotationGuide.html>

```

    iemand een rad voor de ogen draaien -->
6: fixed([[voor,de,ogen],[een,rad],dat],imp_passive),
    b([intransitive,
7: fixed([[quite]],imp_passive),
8: fixed([[quitte]],imp_passive)]))].

```

The example verb tag contains eight fixed complements.⁶ The basic structure of a *fixed* complement is **fixed**([1],2), where position 1 is reserved for the representation of the constituents that are part of the expression, and position 2 is reserved to specify passive information.⁷ Constituents can be either variable or fixed. Variable constituents are represented with predefined labels, see section 2.

Fixed constituents are represented in two ways, depending on their degree of fixedness:

1. A fully *fixed* constituent is represented as one or more components in the full form between square brackets. The components are separated with a comma. The components within the constituent can solely occur as they are represented, thus with the represented form in the represented order. The constituents are realized in the dependency structure with the dependency label *SVP* (Separated Verb Part) and the category label *MWU* (MultiWord Unit). Examples are the complements 5, 6, 7 and 8 in (1): the complements 5 and 6 contain two constituents consisting of two or three fixed components, whereas the complements 7 and 8 contain one constituent with one fixed component.
2. A fully *flexible* constituent is represented as one component in the non-inflected form between round brackets preceded by a predefined label which specifies the dependency of the constituent. The component between the round brackets is the head of the constituent and lexically fixed. The form of the head and the selection of other components, e.g. determiners and modifiers, is flexible. Examples are the complements 1, 2, 3 and 4 in (1).

1.4 Limitations of Alpino

The *MWE lexicon for Dutch* specifies a broad range of MWE properties, which cannot all be converted to Alpino. In this section we list the limitations of the representation of MWEs in Alpino compared to the standard representation, and discuss the decisions taken to handle these limitations.

1. The representation of MWEs in the *MWE Lexicon for Dutch* allows various degrees of fixedness, viz. unmodifiable nouns, limitedly modifiable nouns, and freely modifiable nouns, combined with a free or fixed determiner, whereas the Alpino representation solely distinguishes fully fixed and fully flexible constituents. This means that decisions need to be made about limitedly modifiable nouns and about expressions containing a fixed determiner and a modifiable noun. Although representing an expression as fully fixed yields a more precise representation, one must take into account that each variant of an expression should be represented as a new complement, which may be possible for expressions with a variable determiner, see 5 and 6 in (1), but

⁶The label *part_fixed* is used for fixed expressions headed by a particle verb.

⁷The basic structure of *part_fixed* complements is **part_fixed**(0,[1],2), where position 0 is reserved for the particle.

which is impossible for constituents with a modifiable noun. On the other hand, representing an expression as fully flexible may lead to overgeneralization. Either decision yields undesired results. Since the main objective of the Alpino parser is to have a robust algorithm that always produces some solution, we chose to solely treat constituents with unmodifiable components as fully fixed constituents and to treat all other constituents as fully flexible.

2. The standard representation includes special labels for some determiner types, viz. *INDEF* for indefinite determiners, *PNP* for possessive NPs, and *zijneigen* to refer to possessive pronouns combined with *eigen* ('own'). No corresponding representation exists in Alpino and although it is totally defensible to convert expressions with a special determiner type as fully flexible, i.e. without lexicalizing a determiner, we decided to exclude these expressions from the conversion procedure, so that they can be examined in detail before including them in the Alpino lexicon.
3. A determiner that is represented in the standard representation as the possessive pronoun *zijn* ('his') implies that *zijn* can also take another form such as *mijn* ('my'), *je/jouw* ('your'), etc. In the Alpino lexicon a new complement needs to be created for each variation of this pronoun. An example is the expression *zijn draai vinden* ('find one's way'), which must be encoded with a fixed complement for *mijn draai vinden*, and a fixed complement for *je/jouw draai vinden*, and a fixed complement for *zijn draai vinden*, etc. Furthermore, it is specified in the standard representation whether the pronoun *zijn* is subject bound, object bound, or indirect object bound. It is not possible to indicate boundness in the Alpino lexicon, which means that the sentence *Ik heb zijn draai gevonden* ('I found his way') is not out in Alpino.
4. In the standard lexicon it is explicitly indicated whether an expression must occur in a negative or positive environment. In the Alpino lexicon this feature cannot be specified, which means that we should either abstract away from this property and convert these expressions as the other expressions in the same EC or exclude these expression form the conversion. The latter option has been chosen, so that the expressions can be further analysed before entering them in the Alpino lexicon.
5. As stated in section 1.3, fully fixed constituents can solely occur as they are represented. The information needed to determine the form of the elements that are included in a fully fixed constituent is partly taken from the full form represented in the EXAMPLE-field of the standard representation. It should be noted that full form information for nouns that can be realized in both the singular and the plural form is listed in the EXAMPLE-field with the full form of the singular variant. An example of such an expression is *de breuk lijmen* ('heal the breach'), which can be realized as both *de breuk lijmen* and *de breuken lijmen* ('heal the breaches'). The EXAMPLE of this expression is represented as *hij heeft de breuk gelijmd*.

Alpino requires two complements for these types of expressions: one for the singular form of the noun and one for its plural form. Basically, the complement for the plural form cannot be created, since no plural form can be determined from the standard representation. Although this point seems a limitation of the standard representation it is not: when developing the standard representation we considered this point and decided to represent the singular form in the EXAMPLE-field, assuming that a plural

form of the lexical item is present somewhere in the target lexicon, and can be extracted from there. This is also the case for the Alpino lexicon. The singular and plural form of a noun are lexicalized in *noun.pl*, see (2), which means that we can use this lexicon to extract the plural form of a noun when needed. Lack of time prevent us from applying this step in the current conversion.

```
(2) n([pl(breuken),sg(breuk)],de,[],
      [been,
       bot,
       enkel,
       contract]).
```

2 Conversion procedure

The conversion of the standard representation into the representation required by Alpino consists of a manual part and an automatic part. Before a conversion procedure can be defined, it must be studied which aspects of the standard representation are also included in the Alpino representation. The easiest way to do this is to check which description fields in *patterns.tsv* and *mwes.tsv* are (partly) needed for the conversion:⁸

pattern.tsv

- Unused fields:
 - POS → maintenance field
 - EXAMPLE_MWE → maintenance field
 - EXAMPLE_SENTENCE → maintenance field
- Necessary fields:
 - PATTERN_NAME required to identify instances in *mwes.tsv*.
 - PATTERN and DESCRIPTION required to determine the Alpino pattern. Since the notation of the standard patterns and the notation of the Alpino patterns are both based on a formalization of dependency trees, in particular CGN (*Corpus Gesproken Nederlands* ‘Corpus of Spoken Dutch’) dependency trees (Hoekstra et al., 2003), most labels used in the standard notation can be used in the mapping to Alpino. Exceptions are the labels that indicate the modifiability of the noun and the adjective: *N1* is used for a modifiable noun, *N2* is used for a limitedly modifiable noun, and *A1* is used for a modifiable adjective. See point 1 in section 1.4 for the decisions taken related to this aspect.
 - MAPPING used to identify full form components. Only required when converting to a fully fixed constituent.
 - COMMENTS

mwes.tsv

- Unused fields:
 - EXPRESSION necessary information about the form of the components is extracted from either CL or EXAMPLE.
 - MODIFIER can be used to determine modifiability of a noun, but is irrelevant for the Alpino conversion.
- Necessary fields:
 - PATTERN_NAME required to identify instances of an EC.
 - CL required to identify particle verbs, postpositions, empty determiners, indefinite determiners, possessive NPs, and determiner variation. Also used to determine the non-inflected form of adjectives and nouns in the case of a conversion to a fully flexible constituent.

⁸More information on the description fields can be found in Grégoire (2007b).

- LISTA and LISTB required to identify list verbs.
- EXAMPLE required to identify the full form of adjectives and nouns. Only required when converting to a fully fixed constituent.
- SUBJECT required to lexicalize subject information. Only predefined labels are relevant, but not all labels used in the standard representation can be converted to the Alpino representation. Table 1 shows the labels used in the standard representation and the corresponding Alpino notation.

standard	Alpino
het	het_subj
hetssub	sbar_subj
nohetssub	sbar_subj_no_het
hetvp	vp_subj
pl/sgmet	extra complement with PP-argument headed by <i>met</i> ('with'): pc(met)

Table 1: SUBJECT labels used in the standard representation and the corresponding Alpino notation.

- OBJECT required to lexicalize object information. Table 2 shows the labels used in the standard representation and the corresponding Alpino notation.

standard	Alpino
hetssub	sbar_obj
nohetssub	sbar_obj_no_het
hetvp	vp_obj

Table 2: OBJECT labels used in the standard representation and the corresponding Alpino notation.

- RPRON required to lexicalize pronominalized PP realizations. Table 3 shows the labels used in the standard representation and the corresponding Alpino notation.

standard	Alpino
ssub	er_pp(<i>adposition</i>,X) substitutes the PP-argument, and extra_sbar(X) is added as additional variable.
vp	er_pp(<i>adposition</i>,X) substitutes the PP-argument, and extra_vp(X) is added as additional variable.

Table 3: RPRON labels used in the standard representation and the corresponding Alpino notation.

- CONJUGATION required to determine the correct auxiliary verb in the verb tag.
- POLARITY although polarity cannot be represented in the Alpino lexicon, this field is required in order to exclude MWEs that take the value *NPI* or *PPI* from the conversion.
- COMMENTS commented entries should be checked manually.

Conversion scripts For the implementation three scripts have been created, viz. *template.awk*, *printPatterns.awk* and *alpinclusion.awk*, see appendix A. The software scheme is shown in figure 1 and discussed below.

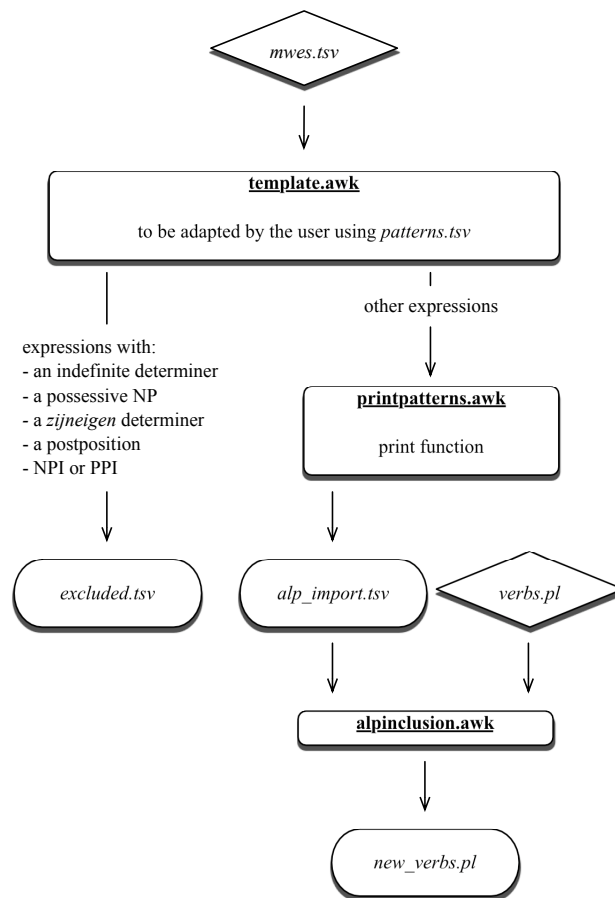


Figure 1: Software scheme

1. The input of *template.awk* is *mwes.tsv*, which file structure is given in appendix A.4.
2. *template.awk* needs to be altered by the user, who needs the following information from *patterns.tsv*:

field 2 PATTERN_NAME

field 4 PATTERN to determine the Alpino pattern

field 5 MAPPING to determine the full form of particular components

field 8 DESCRIPTION to determine the Alpino pattern

The input needed from the user is discussed in step 2 of the manual part of the conversion described below.

Based on the PATTERN_NAME entered by the user an MWE description with this PATTERN_NAME is extracted from *mwes.tsv*. First it is checked whether the description:

- (a) includes an indefinite determiner (INDEF in CL), or
- (b) includes a possessive NP (PNP in CL), or
- (c) includes a *zijneigen* ('his-own') determiner, or

- (d) includes an postposition ([post] in CL), or
- (e) must occur in a negative or positive environment (NPI or PPI in POLARITY).

If the description complies with one of the criteria above it is excluded from the conversion procedure, see section 1.4, and written to *excluded.tsv*. The file structure of *excluded.tsv* is identical to the file structure of *mwes.tsv*.

If the description does not satisfy one of the criteria above, it is further processed. Based on the user input and automatically extracted information from *mwes.tsv*, which includes the target pattern, the verb, conjugation information, subject information, object information, and rpron information, the complements needed by Alpino are printed. This is done by the function *printPattern* in *printPatterns.awk*.

3. *printPatterns.awk* contains the function *printPattern* which writes the information that is needed to include new lexical entries in the Alpino lexicon to the output file *alp-import.tsv*.

The structure of the output file is: *verb <tab> conjugation <tab> complement*.

Examples are given in (3).

- (3)

```
maken h fixed([acc(plan)],norm_passive)
ondernemen h fixed([acc(stap)],norm_passive)
tonen h fixed([[karakter]],norm_passive)
geven h part_fixed(aan,[[de,toon]],norm_passive)
```

Basically, each description processed yields one Alpino complement. In the following cases Alpino requires multiple complements:

- For each variation of the possessive pronoun *zijn* ('one's') a new complement is created, yielding seven extra complements: *mijn, je, jouw, haar, jullie, ons* (or *onze*), *uw*.
- Variation of the determiner, e.g. *de/zijn adem inhouden* ('hold the/one's breath') yields:

```
houden h part_fixed(in,[[de,adem]],norm_passive)
houden h part_fixed(in,[[zijn,adem]],norm_passive)
houden h part_fixed(in,[[mijn,adem]],norm_passive)
houden h part_fixed(in,[[je,adem]],norm_passive)
houden h part_fixed(in,[[jouw,adem]],norm_passive)
houden h part_fixed(in,[[haar,adem]],norm_passive)
houden h part_fixed(in,[[jullie,adem]],norm_passive)
houden h part_fixed(in,[[onze,adem]],norm_passive)
houden h part_fixed(in,[[uw,adem]],norm_passive)
```

- For relevant SUBJECT labels, see table 1. For example EXPRESSION *de moeite lonen* ('to be worthwhile') with SUBJECT *hetvp*, yields

```
lonen h fixed([[de,moeite]],norm_passive)
lonen h fixed([[de,moeite],vp_subj],norm_passive)
```
- For relevant OBJECT labels, see table 2. For example EXPRESSION *iets uit zijn hoofd zetten* ('forget about something') with OBJECT *hetssub*, yields

```
zetten h fixed([[uit,zijn,hoofd],acc],norm_passive)
zetten h fixed([[uit,het,hoofd],sbar_obj],norm_passive)
```

- For relevant RPRON labels, see table 3. For example EXPRESSION *de moeite lonen* with RPRON *hetvp*, yields

```
lonen h fixed([[de,moeite]],norm_passive)
lonen h fixed([[de,moeite],vp_subj],norm_passive)
```

- For multiple list verbs. For each verb in LISTA and LISTB a new complement is created.
- For obligatory variable PP-arguments: the possibility of pronominalization of the complement of a PP-argument needs to be lexicalized, see table 4.

```
uiten h fixed([acc(kritiek),pc(op)],norm_passive)
--> kritiek uit en op
uiten h fixed([acc(kritiek),er_pp(op)],norm_passive)
--> kritiek er op uit en
```

- For obligatory variable locative/directional complements: locative adverbs and locative/directional PP constituents need to be represented separately:

```
vinden h fixed([[aansluiting],ld_adv],norm_passive)
--> ergens aansluiting vinden
vinden h fixed([[aansluiting],ld_pp],norm_passive)
--> aansluiting vinden in/bij/etc. ...
```

4. *alpinclusion.awk* takes as input *alp_import.tsv* and *verbs.pl*. Recall that the structure of the file is: *verb <tab> conjugation <tab> complement*. The script simply inserts the *complement* in the verb tag *verb* under the right *conjugation*. The script ignores *fixed* and *part-fixed* complements that are already present in *verbs.pl*, which means that the output file may contain duplicate complements. The output is written to *new_verbs.pl*.

Input example from *alp_import.tsv*:

```
lonen h fixed([[de,moeite],vp_subj],norm_passive)
lonen h fixed([[de,moeite]],norm_passive)
```

Verb tag for *lonen* ('be worth') in *verbs.pl*:

```
v(loon,loont,lonen,geloond,loonde,loonden,
  [h([np_np,
intransitive,
transitive,
vp_subj_np]))]).
```

Output example in *new_verbs.pl*:

```
v(loon,loont,lonen,geloond,loonde,loonden,
  [h([np_np,
fixed([[de,moeite]],norm_passive),
```

```
fixed([[de,moeite],vp_subj],norm_passive),
intransitive,
transitive,
vp_subj_np]])).
```

Manual part Having the three scripts, the manual part of the conversion consists of the following steps:

1. Use a copy of *template.awk* and rename it to the EC that needs to be converted.
2. Take one instance *X* of the chosen EC from *mwes.tsv* and use its CL-field and EXAMPLE-field to adapt the template by following the guidelines indicated with -->:
 - # --> Fill in the equivalence class:
PATTERN_NAME = "ec"
 - # --> Determine the Target Pattern (TP), although it should be initiated at the end of the script it is useful to create it at this stage.
The TP must be determined manually on the basis of the PATTERN and the DESCRIPTION (in patterns.tsv), and knowledge of the Alpino Lexicon.
Use obvious names for the variables, e.g. determiner for a determiner component, noun for a noun components, etc.
Since passive information is not included in the standard representation, norm_passive is used for all complements.
An example of a template TP is:
fixed([[VARIABLE1,VARIABLE2],variable_constituent],norm_passive)
Concrete examples of a TP are:
fixed([[determiner,noun]],norm_passive)
fixed([[determiner,noun],acc],norm_passive)
fixed([[determiner,noun],pc(adposition)],norm_passive)

Variable constituents are represented in Alpino with predefined labels. Table 4 shows the standard notation of variable constituents and the corresponding Alpino labels.

standard	Alpino
[.hd:REFLV (<i>index</i>)]	refl
[.obj1:NP (var)]	acc
[.obj2:NP (var)]	dat
[.pc:PP [.hd (1)] [.obj1:NP (var)]]	pc(1) and er_pp(1)
[.ld (var)]	ld_pp and ld_adv
[.vc:OTI (var)]	vp
[.vc:TI (var)]	vp
[.vc:SSUB (var)]	sbar

Table 4: The standard notation of variable constituents and the corresponding Alpino labels.

- # --> Determine the components that need to be substituted for the

variables in the TP.

NOTE1: determiner values, adpositions and non-inflected nouns should be extracted from the CL (stored in the "comp"-array) and full forms of other values should be extracted from the EXAMPLE (stored in the "exam"-array), using the MAPPING.

```
VARIABLE1 = comp[POSITION] or exam[POSITION]
VARIABLE2 = comp[POSITION] or exam[POSITION]
```

The elements in CL are separated by a space. The string is split and the first element is saved in comp[1], the second element is saved in comp[2], etc. For example if CL of one the instances of the chosen EC is *de moeite[sg] lonen*, then comp[1] = *de*, comp[2] = *moeite[sg]* and comp[3] = *lonen*. The corresponding EXAMPLE is *iets heeft de moeite geloond*, splitting the string yields: exam[1] = *iets*, exam[2] = *heeft*, exam[3] = *de*, exam[4] = *moeite*, and exam[5] = *geloond*.

FULLY FIXED

Take into account the gender, diminutive and number information of the noun to avoid 'ons'/'onze' errors, if the determiner is 'zijn'.

```
if(determiner ~ /zijn/ && ((comp[POSITION] ~ /het/
|| comp[POSITION] ~ /dim/) && comp[POSITION] !~ /pl/)
  sub(/zijn/,"ons",determiner)
else sub(/zijn/,"onze",determiner)
```

The form of the 3rd person plural possessive pronoun depends on the form of the noun and is either *ons* when the gender of the noun is feminine or masculine or *onze* when the gender of the noun is neutral or when the noun is singular diminutive. The right form is chosen in this step.

To set a value to a noun, empty determiners need to be taken into account.
Therefore use this code to determine the full form of the noun.

```
if(determiner == "EMP")   noun = exam[POSITION-1]
else   noun = exam[POSITION]
```

FULLY FLEXIBLE: adjectives and nouns extracted from the "comp"-array should be separated from their parameters

```
temp = comp[POSITION]

if(match(temp,/[\/])   VARIABLE = substr(temp,1,RSTART-1)
else VARIABLE = temp
```

As stated in section 1.3, a fully flexible constituent is represented as one component in the non-inflected form between round brackets preceded by a predefined

label which specifies the dependency of the constituent. Table 5 shows the standard notation of dependency labels of flexible constituents and the corresponding Alpino labels that are used in the conversion.

standard	Alpino
obj1	acc
predc:NP	np_pred
predc:AP	ap_pred

Table 5: Dependency labels used in the standard representation and the corresponding Alpino notation.

- # --> Check which POSITION the head of the expression occupies in the CL (comp-array), and set its value.
If the head is a list of verbs, set the value to "list"
hd = comp[POSITION] or hd = "list"
 - # --> Fill in the TP in a sprintf() environment
TP = sprintf("fixed([[%s,%s]],norm_passive)",VARIABLE1, VARIABLE2)
3. Run the script with the following command line:
`awk -f template.awk -f printPatterns.awk mwes.tsv`
 4. Check the output file *alp_import.tsv* for possible errors. Errors may occur either due to mistakes in *mwes.tsv*, or due to phenomena that are overlooked and not included in the conversion tool. Limitations of the tool that should be corrected manually are:
 - the incorrect conversion of an expression with two determiner positions that both contain alternating components, e.g. *met de/zijn hand over het/zijn hart strijken*. The conversion script solely deals with the first alternating determiner position, which means that extra complements should be created manually for the second alternating position.
 - the Alpino convention that words that are capitalized should be between single quotes, e.g. *Nederlandse* should be manually corrected to '*Nederlandse*'.

Make sure *alp_import.tsv* uses character encoding Latin-1 and Unix convention (LF) for newline.

5. *new_verbs.pl* is created by using the following command line:
`awk -f alpinclusion.awk alp_import.tsv`

Automatic part Step three of the manual conversion part leads to an automatic conversion of all instances in *mwes.tsv* which PATTERN_NAME correspond to the PATTERN_NAME specified in *template.awk*. As stated, the output, in the format *verb <tab> conjugation <tab> complement*, is written to *alp_import.tsv*.

Step four of the manual conversion part automatically incorporates the created complements in *alp_import.tsv* into *verbs.pl*, yielding the file *new_verbs.pl*. An illustration of the conversion procedure is given in the next section.

3 Example conversions

This section illustrates the conversion procedure as described in the previous section for EC1-EC5. For each conversion it is specified which information is taken from *patterns.tsv* and *mwes.tsv*. Furthermore, the second step of the manual part is talked through, and examples of *alp_import.tsv* are given.

3.1 EC1

Information extracted from *patterns.tsv*:

- PATTERN_NAME ec1
- PATTERN [.VP [.obj1:NP [.det:D (1)] [.hd:N (2)]] [.hd:V (3)]]
- DESCRIPTION Expressions headed by a verb, taking a direct object consisting of a fixed determiner and an unmodifiable noun.
- MAPPING 3 4 5

Information extracted from a random instance of *ec1* in *mwes.tsv*:

- CL zijn kans[pl] waar_nemen[part]
- EXAMPLE hij heeft zijn kansen waargenomen

Manual part Follow the guidelines in the template, indicated with `-->`:

- # --> Fill in the equivalence class:
PATTERN_NAME = "ec1"

- # --> Determine the Target Pattern (TP):

The pattern is headed by a verb, which takes one complement consisting of a determiner and an unmodifiable noun. Since the noun is unmodifiable, the pattern is interpreted as fully fixed, and accordingly converted to the Alpino representation: *fixed([[**determiner**,**noun**]],norm_passive)*

- # --> Determine the components that need to be substituted for the variables in the TP.

The TP contains two variables: **determiner** and **noun**.

determiner is extracted from the component list. From the information in PATTERN *[.det:D (1)]*, the position of the determiner in CL can be determined: the index is 1, which means that the determiner occupies the first position in CL. Accordingly **determiner** should be assigned the value that is stored in the first position of the *comp*-array: *determiner = comp[1]*

noun is extracted from EXAMPLE, since the full form is needed. To determine the position of the noun in EXAMPLE, information stored in MAPPING is used. From the information in PATTERN *[.hd:N (2)]*, the position of the noun in CL can be determined: the index is 2, which means that the noun occupies the second

position in CL. MAPPING denotes that the component that occupies the second position in CL, occupies the fourth position in EXAMPLE:

```
if(determiner == "EMP") noun = exam[4-1]
else noun = exam[4]
```

- # --> Check which POSITION the head of the expression occupies in the CL (comp), and set its value.
If the head is a list of verbs, set the value to "list"

From the information in PATTERN *[.hd:V (3)]*, the position of the head in CL can be determined: the index is 3, which means that the head occupies the third position in CL: *hd = comp[3]*

- # --> Fill in the TP

The TP was determined above, and must be encoded in a *sprintf* environment:

```
TP = sprintf("fixed([[%s,%s]],norm_passive)",determiner, noun)
```

***alp_import* examples** Note that if the determiner is the possessive pronoun *zijn*, additional complements for each variant of *zijn* are created automatically. Expressions containing a particle verb are automatically recognized and converted as a *part_fixed* complement (see the last example below).

```
lonen h fixed([[de,moeite],vp_subj],norm_passive)
lonen h fixed([[de,moeite]],norm_passive)
krijgen h fixed([[gelijk]],norm_passive)
krijgen h fixed([[zijn,gelijk]],norm_passive)
krijgen h fixed([[mijn,gelijk]],norm_passive)
krijgen h fixed([[je,gelijk]],norm_passive)
krijgen h fixed([[jouw,gelijk]],norm_passive)
krijgen h fixed([[haar,gelijk]],norm_passive)
krijgen h fixed([[jullie,gelijk]],norm_passive)
krijgen h fixed([[ons,gelijk]],norm_passive)
krijgen h fixed([[uw,gelijk]],norm_passive)
geven h fixed([[de,doorslag],sbar_subj],norm_passive)
geven h fixed([[de,doorslag],sbar_subj_no_het],norm_passive)
geven h fixed([[de,doorslag]],norm_passive)
geven h part_fixed(aan,[[de,toon]],norm_passive)
```

3.2 EC2

Information extracted from *patterns.tsv*:

- PATTERN_NAME ec2
- PATTERN *[.VP [.obj1:NP [.hd:N1 (1)]] [.hd:V (list)]]*
- DESCRIPTION Expressions headed by a verb, taking a direct object consisting of a modifiable noun (list).
- MAPPING *not needed, since converting to fully flexible constituent.*

Information extracted from a random instance of *ec2* in *mwes.tsv*:

- CL blunder
- EXAMPLE *not needed, since converting to fully flexible constituent.*

Manual part Follow the guidelines in the template, indicated with `-->`:

- # --> Fill in the equivalence class:

```
PATTERN_NAME = "ec2"
```

- # --> Determine the Target Pattern (TP):

The pattern is headed by a verb, which takes one complement (a direct object) consisting of a modifiable noun. Since the noun is modifiable, the pattern is interpreted as fully flexible, and accordingly converted to the Alpino representation:
*fixed([acc(**noun**)],norm_passive)*

- # --> Determine the components that need to be substituted for the variables in the TP.

The TP contains one variable: **noun**.

noun is extracted from CL, since the non-inflected form is needed. From the information in PATTERN *[.hd:N1 (1)]*, the position of the noun in CL can be determined: the index is 1, which means that the noun occupies the first position in CL. So, the value is set to: *comp[1]*

- # --> Check which POSITION the head of the expression occupies in the CL (comp), and set its value.

```
# If the head is a list of verbs, set the value to "list"
```

From the information in PATTERN *[.hd:V (list)]*, the value can be determined:
hd = "list"

- # --> Fill in the TP

The TP was determined above, and must be encoded in a *sprintf* environment:

```
TP = sprintf("fixed([acc(%s)],norm_passive)",noun)
```

alp_import examples

```
wagen h fixed([acc(poging)],norm_passive)
nemen h part_fixed(onder,[acc(poging)],norm_passive)
doen h fixed([acc(poging)],norm_passive)
trekken h fixed([acc(conclusie)],norm_passive)
nemen h fixed([acc(stap)],norm_passive)
```

3.3 EC3

Information extracted from *patterns.tsv*:

- PATTERN_NAME ec3
- PATTERN [.VP [.obj1:NP [.det:D (1)] [.hd:N2 (2)]] [.hd:V (3)]]
- DESCRIPTION Expressions headed by a verb, taking a direct object consisting of a fixed determiner and a limited modifiable noun.
- MAPPING *not needed, since converting to fully flexible constituent.*

Information extracted from a random instance of *ec3* in *mwes.tsv*:

- CL de rollen omdraaien
- EXAMPLE *not needed, since converting to fully flexible constituent.*

Manual part Follow the guidelines in the template, indicated with `-->`:

- # --> Fill in the equivalence class:

PATTERN_NAME = "ec3"

- # --> Determine the Target Pattern (TP):

The pattern is headed by a verb, which takes one complement (a direct object) consisting of a determiner and a limitedly modifiable noun. Since the noun is limitedly modifiable, the pattern is interpreted as fully flexible, and accordingly converted to the Alpino representation: *fixed([acc(**noun**)],norm_passive)*

- # --> Determine the components that need to be substituted for the variables in the TP.

The TP contains one variable: *noun*.

noun is extracted from CL, since the non-inflected form is needed. From the information in PATTERN *[.hd:N2 (2)]*, the position of the noun in CL can be determined: the index is 2, which means that the noun occupies the second position in CL. So, the value is set to: *comp[2]*

- # --> Check which POSITION the head of the expression occupies in the CL (comp), and set its value.

If the head is a list of verbs, set the value to "list"

From the information in PATTERN *[.hd:V (3)]*, the position of the head in CL can be determined: the index is 3, which means that the head occupies the third position in CL: *hd = comp[3]*

- # --> Fill in the TP

The TP was determined above, and must be encoded in a *sprintf* environment:

*TP = sprintf("fixed([acc(%s)],norm_passive)",**noun**)*

alp_import examples

```
draaien h part_fixed(om,[acc(rol)],norm_passive)
verdelen h fixed([acc(taak)],norm_passive)
doen h fixed([acc(best)],norm_passive)
wagen h fixed([acc(stap)],norm_passive)
doen h fixed([acc(aangifte)],norm_passive)
```

3.4 EC4

Information extracted from *patterns.tsv*:

- PATTERN_NAME ec4
- PATTERN [.VP [.obj2:NP (var)] [.obj1:N1 [.hd:N (1)]] [.hd:V (list)]]
- DESCRIPTION Expressions headed by a verb, taking (1) a variable indirect object, and (2) a direct object consisting of a modifiable noun (list).
- MAPPING *not needed, since converting to fully flexible constituent.*

Information extracted from a random instance of *ec4* in *mwes.tsv*:

- CL vraag
- EXAMPLE *not needed, since converting to fully flexible constituent.*

Manual part Follow the guidelines in the template, indicated with -->:

- # --> Fill in the equivalence class:
PATTERN_NAME = "ec4"
- # --> Determine the Target Pattern (TP):
*fixed([acc(**noun**),dat],norm_passive)*
- # --> Determine the components that need to be substituted for the variables in the TP.
The TP contains one variable: *noun = comp[2]*
- # --> Check which POSITION the head of the expression occupies in the CL (comp), and set its value.
If the head is a list of verbs, set the value to "list"
hd = "list"
- # --> Fill in the TP
*TP = sprintf("fixed([acc(%s),dat],norm_passive)",**noun**)*

alp_import examples

```
boezemen h part_fixed(in, [acc(angst), dat, sbar_subj], norm_passive)
boezemen h part_fixed(in, [acc(angst), dat], norm_passive)
geven h fixed([acc(compliment), dat], norm_passive)
doen h fixed([acc(belofte), dat], norm_passive)
doen h fixed([acc(verdriet), dat], norm_passive)
bezorgen h fixed([acc(baan), dat], norm_passive)
```

3.5 EC5

Information extracted from *patterns.tsv*:

- PATTERN_NAME ec5
- PATTERN [.VP [.obj1:NP [.hd:N1 (1)]] [.hd:V (list)]]
- DESCRIPTION Expressions headed by a verb, taking a direct object consisting of a modifiable noun (list).
- MAPPING *not needed, since converting to fully flexible constituent.*
- COMMENTS The nouns in this class behave both like count nouns (as they can both be plural and singular) and as mass nouns (as they can take an empty determiner and indefinites such as 'veel' and 'weinig').

Information extracted from a random instance of *ec5* in *mwes.tsv*:

- CL actie
- EXAMPLE *not needed, since converting to fully flexible constituent.*

Manual part The important part in the comments is that the nouns in this EC can take an empty determiner. Since determinerless NPs are not allowed in standard grammar when the head of the NP is a singular count noun, we need to adapt the template file so that it creates an additional complement for singular count nouns in this class. The basic TP is *fixed([acc(**noun**)], norm_passive)*, if the noun is singular count – which can be determined by checking the parameters attached to noun – an additional complement is created with a fully fixed constituent containing the singular count noun: *fixed([[**noun**]], norm_passive)*, see *ec5count.awk*.

Follow the guidelines in the template, indicated with -->:

- # --> Fill in the equivalence class:
PATTERN_NAME = "ec5"
- # --> Determine the Target Pattern (TP):
*ec5.awk: fixed([acc(**noun**)], norm_passive)*
*ec5count.awk: fixed([[**noun**]], norm_passive)*

- # --> Determine the components that need to be substituted for the variables in the TP.
The TP contains one variable: *noun = comp[2]*
In *ec5count.awk* the parameters are checked: if CL does NOT contain *[pl]* or *[mass]* it is further processed.
- # --> Check which POSITION the head of the expression occupies in the CL (comp), and set its value.
If the head is a list of verbs, set the value to "list"
hd = "list"
- # --> Fill in the TP
ec5.awk: TP = sprintf("fixed([acc(%s)],norm_passive)",noun)
ec5count.awk: TP = sprintf("fixed([[%s]],norm_passive)",noun)

ec5_import examples

```
maken h fixed([acc(indruk)],norm_passive)
sorteren h fixed([acc(effect)],norm_passive)
hebben h fixed([acc(effect)],norm_passive)
verrichten h fixed([acc(onderzoek)],norm_passive)
doen h fixed([acc(onderzoek)],norm_passive)
lopen h fixed([acc(risico)],norm_passive)
voeren h fixed([acc(actie)],norm_passive)
lijden h fixed([acc(verlies)],norm_passive)
```

ec5count_import examples

```
maken h fixed([[indruk]],norm_passive)
sorteren h fixed([[effect]],norm_passive)
hebben h fixed([[effect]],norm_passive)
verrichten h fixed([[onderzoek]],norm_passive)
doen h fixed([[onderzoek]],norm_passive)
lopen h fixed([[risico]],norm_passive)
```

3.6 EC6

Information extracted from *patterns.tsv*:

- PATTERN_NAME ec6
- PATTERN [.VP [.ld (var)] [.obj1:NP [.det:D (1)] [.hd:N (2)]] [.hd:V (3)]]
- DESCRIPTION Expressions headed by a verb, taking (1) a variable locative/directional complement, and (2) a direct object consisting of a fixed determiner and an unmodifiable noun.
- MAPPING 4 5 6

Information extracted from a random instance of *ec6* in *mwes.tsv*:

- CL zijn einde vinden
- EXAMPLE hij heeft ergens zijn einde gevonden

Manual part Follow the guidelines in the template, indicated with `-->`:

- # --> Fill in the equivalence class:
`PATTERN_NAME = "ec6"`
- # --> Determine the Target Pattern (TP):
 The variable locative/directional complement is represented with *ld_pp*. A complement with the alternative representation *ld_adv* will be created automatically:
`fixed([[determiner,noun],ld_pp],norm_passive)`
- # --> Determine the components that need to be substituted for the variables in the TP.
`determiner = comp[1]`

`if(determiner == "EMP") noun = exam[5-1]`
`else noun = exam[5]`
- # --> Check which POSITION the head of the expression occupies in the CL (comp), and set its value.
`hd = comp[3]`
- # --> Fill in the TP
 The TP was determined above, and must be encoded in a *sprintf* environment:
`TP = sprintf("fixed([[%s,%s],ld_pp],norm_passive)",determiner, noun)`

alp_import examples

```
nemen h fixed([[een,kijkje],ld_adv],norm_passive)
nemen h fixed([[een,kijkje],ld_pp],norm_passive)
zwaaien h fixed([[de,scepter],ld_adv],norm_passive)
zwaaien h fixed([[de,scepter],ld_pp],norm_passive)
```

References

- Grégoire, N. (2007a), Design and implementation of a lexicon of Dutch multiword expressions, *Proceedings of the Workshop on A Broader Perspective on Multiword Expressions*, Association for Computational Linguistics, Prague, Czech Republic, pp. 17–24.
- Grégoire, N. (2007b), MWE lexicon for Dutch: Encoding protocol, *Technical report*, STEVIN IRME.
- Hoekstra, H., Moortgat, M., Renmans, B., Schouppe, M., Schuurman, I. and van der Wouden, T. (2003), CGN syntactische annotatie.
- Odiijk, J. (2003), Towards a standard for multi-word expressions. ISLE Project Report.

A Conversion scripts and input information

A.1 template.awk

```
# AWK script - August 2007 - STEVIN IRME
# Nicole Gregoire - Nicole.Gregoire@let.uu.nl
# Template to convert the MWE lexicon for Dutch to the Alpino Lexicon
# Command line: $ awk -f template.awk -f printPatterns.awk mwes.tsv
# More information can be found in the document Alpino_conversion.pdf

#####

# Files needed:
# mwes.tsv           Export file with MWE descriptions
# printPatterns.awk Awk script with print functions

# Information needed from patterns.tsv:
# PATTERN_NAME
# PATTERN and DESCRIPTION to determine Alpino pattern
# MAPPING to determine the full form of particular components

#####

# Abstract:
# This template should be used to convert equivalence classes from the MWE Lexicon
# for Dutch to the Alpino Lexicon.
# The template forms the basis of the manual part of the conversion,
# i.e. the user needs to specify information required by the automatic conversion
# procedure here.

# MWE entries which cannot be implemented in Alpino are saved in "excluded.tsv"
# MWE entries which can be implemented in Alpino are further processed and send to
# the function printPattern in printPatterns.awk

#####

## IMPORTANT INFORMATION FOR THE USER ##

# An arrow (-->) indicates that input is needed from the user.

# Please check carefully that all values are entered.

#####

BEGIN{
FS = "\t" # Fields are separated by a tab
RS = "\n" # Records are separated by a blank line
```

```

# --> Fill in the equivalence class:
PATTERN_NAME = "ec"

# --> Determine the Target Pattern (TP), although it should be initiated at the end of
# the script it is useful to create it at this stage.
# The TP must be determined manually on the basis of the PATTERN and the DESCRIPTION
# (in patterns.tsv), and knowledge of the Alpino Lexicon.
# Use obvious names for the variables, e.g. determiner for a determiner component, noun
# for a noun components, etc.
# Since passive information is not included in the standard representation, norm_passive
# is used for all complements.
# Example template: fixed([[VARIABLE1,VARIABLE2],variable_constituent],norm_passive)
# Concrete examples of a TP are:
# fixed([[determiner,noun],acc],norm_passive)
# fixed([[determiner,noun],pc(adposition)],norm_passive)
# fixed([[determiner,noun],variable_constituent],norm_passive)

}

{
EXAMPLE = ""

# Check whether PATTERN_NAME1 - PATTERN_NAME4 corresponds to PATTERN_NAME and save
# corresponding EXAMPLE
if($2 == PATTERN_NAME) EXAMPLE = $13
else if($3 == PATTERN_NAME) EXAMPLE = $14
else if($4 == PATTERN_NAME) EXAMPLE = $15
else if($5 == PATTERN_NAME) EXAMPLE = $16

# If EXAMPLE is assigned a value,
if(EXAMPLE != "")
{
# Get Component List (CL) form mwes.tsv
CL = $6

# Expressions with an indefinite determiner or a possessive pronoun as determiner
# are excluded from the conversion procedure
if(match(CL,/PNP/) || match(CL,/INDEF/) || match(CL,/zijneigen/)
|| match(CL,/\[post\]/)) print $0 >> "excluded.tsv"
else if(match($17,/PPI/) || match($17,/NPI/)) print $0 >> "excluded.tsv"
else {

# The elements in CL are separated by a space. The string is split and the
# first element is saved in comp[1], the second element is saved in comp[2], etc.
var1 = split(CL, comp, " ")

# The elements in EXAMPLE are separated by a space. The string is split and the

```

```

# first element is saved in exam[1], the second element is saved in exam[2], etc.
var2 = split(EXAMPLE, exam, " ")

# --> Determine the components that need to be substituted for the variables
# in the TP.
# NOTE1: determiner values, adpositions and non-inflected nouns should be
# extracted from the CL (stored in the "comp"-array) and full forms of other
# values should be extracted from the EXAMPLE (stored in the "exam"-array),
# using the MAPPING.

VARIABLE1 = comp[POSITION] or exam[POSITION]
VARIABLE2 = comp[POSITION] or exam[POSITION]

# FULLY FIXED
# Take into account the gender information of the noun to avoid 'ons'/'onze'
# errors, if the determiner is 'zijn'.
if(determiner ~ /zijn/ && ((comp[POSITION] ~ /het/ || comp[POSITION] ~ /dim/)
&& comp[POSITION] !~ /pl/)) sub(/zijn/,"ons",determiner)
else sub(/zijn/,"onze",determiner)

# To set a value to a noun, empty determiners need to be taken into account.
# Therefore use this code to determine the full form of the noun.
if(determiner == "EMP") noun = exam[POSITION-1]
else noun = exam[POSITION]

# FULLY FLEXIBLE: adjectives and nouns extracted from the "comp"-array should
# be separated from their parameters
temp = comp[POSITION]

if(match(temp,/[/]) VARIABLE = substr(temp,1,RSTART-1)
else VARIABLE = temp

# --> Check which POSITION the head of the expression occupies in the CL
# (comp-array), and set its value.
# If the head is a list of verbs, set the value to "list"
hd = comp[POSITION] or hd = "list"

# --> Fill in the TP in a sprintf() environment
TP = sprintf("fixed([[s,%s]],norm_passive)",VARIABLE1, VARIABLE2)

##### --> Final check whether all user input is given. #####

# Get LISTA mwes.tsv
LISTA = $7

# Get LISTB mwes.tsv
LISTB = $8

```

```

# Get CONJUGATION mwes.tsv
CONJUGATION = $16

# Get SUBJECT from mwes.tsv
SUBJECT = $9
var3 = split(SUBJECT, subj, " ") # Only the predefined labels are needed,
# which are separated from the actual subject by a space

# Get OBJECT from mwes.tsv
OBJECT = $10
var4 = split(OBJECT, obj, " ") # Only the predefined labels are needed,
# which are separated from the actual object by a space

# Get RPRON from mwes.tsv
RPRON = $11

# Ready to print the information that is needed by Alpino on the basis of
# information gathered in this script:
# the target pattern, the verb, conjugation information, subject information,
# object information, and rpron information.
printPattern(TP,hd,LISTA,LISTB,CONJUGATION,subj[1],obj[1],RPRON)
}
}
}

```

A.2 printPatterns.awk

```
# AWK script - July 2007 - STEVIN IRME
# Nicole Gregoire - Nicole.Gregoire@let.uu.nl
# Function for printing Alpino patterns on the basis of information gathered in
# template.awk
# The fuction is called from template.awk
# More information can be found in the document Alpino_conversion.pdf

#####

function printPattern(PATTERN, HEAD, LIA, LIB, CONJ, SUBJ, OBJ, RPRONOM) {

    headarr = 1 # variable used for heads array
    delete heads # empty the array
    delete lista
    delete listb

    # Alpino requires a new complement for each variation of an expression.
    # Expressions with a possessive pronoun yield a separate complement for each pronoun.
    pron[1] = "mijn"
    pron[2] = "je"
    pron[3] = "jouw"
    pron[4] = "haar"
    pron[5] = "jullie"
    pron[6] = "zijn"
    pron[7] = "uw"

    # If the head of the expression is a list of verbs, then the verbs are stored in
    # array 'heads'
    if(match(HEAD,"list"))
    {
        if(LIA != "-") vartemp1 = split(LIA, lista, " ")
        if(LIB != "-") vartemp2 = split(LIB, listb, " ")

        for(k in lista)
        {
            heads[headarr] = lista[k]
            headarr++
        }

        for(l in listb)
        {
            if(!match(listb[l],/\(/))
            {
                heads[headarr] = listb[l]
                headarr++
            }
        }
    }
}
```

```

    }
}
else heads[headarr] = HEAD

# Loop through array 'heads'
for(m in heads)
{
  detvar = 0 # set trigger for determiner variation
  a = 1 # variable used for patterns array
  c = 1 # variable used for patterns2 array
  delete patterns
  delete patterns2
  addToPat = ""
  tmp_pattern = PATTERN

  # If the head of the expression is a particle verb, TP changes from fixed
  # to part_fixed
  if(match(heads[m],/[part\]/))
  {
    partverb = substr(heads[m],1,RSTART - 1) # Delete parameter [part], to
    remain the particle_verb.
    match(partverb,"_")
    part = substr(partverb,1,RSTART - 1)
    verb = substr(partverb,RSTART + 1)

    temp = "(" part ","
    if(sub(/\(/,temp,tmp_pattern) == 1){}
    else tmp_pattern = tmp_pattern "(" part ")"
  }

# The print form is verb conjugation pattern
  firstpart = verb "\t" CONJ "\tpart_"
}
else
{
  firstpart = heads[m] "\t" CONJ "\t"
}

# Check whether the pattern contains alternating determiners.
# This is indicated with a slash.
# If a slash is found, multiple patterns are created and stored in
# the array 'patterns'
if(match(tmp_pattern,[a-zA-Z]*\/[\/a-zA-Z]*\/))
{
  determiners = substr(tmp_pattern,RSTART,RLENGTH)
  var3 = split(determiners, det, "/")

  for(b in det)
  {

```

```

        extra_pat = tmp_pattern
        sub(determiners,det[b],extra_pat)
        patterns[a] = extra_pat
        a++
    }
    detvar = 1;
}

# If there is no variation of the determiner, the pattern does not change
# and is stored in the array
if(detvar == 0)    patterns[a] = tmp_pattern

# If relevant SUBJECT, OBJECT or RPRON information is present, then additional
# patterns need to be created.
# A new array 'patterns2' is created: both items from 'patterns' and newly
# created patterns are stored in the array.
# Since solely one variable constituent can be present in an complement, it
# is possible to check the labels linearly.
for(d in patterns)
{
    patterns2[c] = patterns[d]
    c++

    #create extra pattern for patterns with 'ld_pp', i.e. replace 'ld_pp'
    # with 'ld_adv'
    if(patterns[d] ~ /ld\_pp/)
    {
        extra_pat = patterns[d]
        sub(/ld\_pp/, "ld_adv", extra_pat)
        patterns2[c] = extra_pat
        c++
    }

    #create extra pattern for patterns with 'pc(adposition)', i.e. replace
    # 'pc(adposition)' with 'er_pp(adposition)'
    if(patterns[d] ~ /pc\(/)
    {
        extra_pat = patterns[d]
        sub(/pc\(/, "er_pp(", extra_pat)
        patterns2[c] = extra_pat
        c++
    }
}

if(SUBJ ~ /\[\]/)
{
    # Variable object and subject constituents are added right
    # before the last ].
    match(patterns[d], /\]/)
}

```

```

temp = RSTART
match(substr(patterns[d],temp+1),/\[/])

first = substr(patterns[d], 1, RSTART+temp-1)
second = substr(patterns[d],RSTART+temp)

if(SUBJ ~ /\[het\]/)
{
    patterns2[c] = first ",het_subj" second
    c++
}
if(SUBJ ~ /\[hetssub\]/)
{
    patterns2[c] = first ",sbar_subj" second
    c++
}
if(SUBJ ~ /\[nohetssub\]/)
{
    patterns2[c] = first ",sbar_subj_no_het" second
    c++
}
if(SUBJ ~ /\[hetvp\]/)
{
    patterns2[c] = first ",vp_subj" second
    c++
}
if(SUBJ ~ /sgmet/)
{
    patterns2[c] = first ",pc(met)" second
    c++
}
}

if(OBJ ~ /[\[]/)
{
    if(OBJ ~ /\[hetssub\]/)
    {
        extra_pat = patterns[d]
        sub(/acc/,"sbar_obj",extra_pat)
        patterns2[c] = extra_pat
        c++
    }
    if(OBJ ~ /\[nohetssub\]/)
    {
        extra_pat = patterns[d]
        sub(/acc/,"sbar_obj_no_het",extra_pat)
        patterns2[c] = extra_pat
        c++
    }
}

```

```

    }
    if(OBJ ~ /\[hetvp\]/)
    {
        extra_pat = patterns[d]
        sub(/acc/,"vp_obj",extra_pat)
        patterns2[c] = extra_pat
        c++
    }
}

if(RPRONOM ~ /\[[]/)
{
    extra_pat = patterns[d]
    sub(/pc\(/,"er_pp(",extra_pat)

    match(extra_pat,/\)\)/)
    temp = RSTART
    first = substr(extra_pat, 1, RSTART-1)
    second = substr(extra_pat,RSTART+1)

    if(RPRONOM ~ /\[ssub\]/)
    {
        patterns2[c] = first ",X),extra_sbar(X)" second
        c++
    }

    if(RPRONOM ~ /\[vp\]/)
    {
        patterns2[c] = first ",X),extra_vp(X)" second
        c++
    }
}

}
# Check all items in the array 'patterns2'
for(e in patterns2)
{
    # If the pattern matches EMP, then EMP is deleted and the pattern is
    # printed to "alp_import.tsv"
    if(match(patterns2[e],/EMP\,/))
    {
        extra_pat = patterns2[e]
        gsub(/EMP\,/,"",extra_pat)
        print firstpart extra_pat >> "alp_import.tsv"
    } # else the pattern is printed to "alp_import.tsv"
    else print firstpart patterns2[e] >> "alp_import.tsv"

    # If the pattern matches 'ons' or 'onze', then extra patterns
    # are created for each form of 'zijn' and printed to "alp_import.tsv"

```

```

if(match(patterns2[e],/\,ons\,/) || match(patterns2[e],/\[ons\,/)
|| match(patterns2[e],/\,onze\,/) || match(patterns2[e],/\[onze\,/)
{
  for(f=1;f<=7;f++)
  {
    extra_pat = patterns2[e]
    if(match(patterns2[e],/\,ons\,/) || match(patterns2[e],/\[ons\,/)
      sub(/ons/,pron[f],extra_pat)
    if(match(patterns2[e],/\,onze\,/) || match(patterns2[e],/\[onze\,/)
      sub(/onze/,pron[f],extra_pat)
    sub(/EMP\,/, "",extra_pat)
    print firstpart extra_pat >> "alp_import.tsv"
  }
}
}

return
}

```

A.3 alpinclusion.awk

```
# AWK script - August 2007 - STEVIN IRME
# Nicole Gregoire - Nicole.Gregoire@let.uu.nl
# Template to convert the MWE lexicon for Dutch to the Alpino Lexicon
# Command line: $ awk -f alpinclusion.awk alp_import.tsv
# More information can be found in the document Alpino_conversion.pdf

#####

# Files needed:
# alp_import.tsv      Output from template.awk
# verbs.pl           Alpino verb lexicon

#####

# Abstract:
# This file is used to add the output of printPatterns.awk, written to alp_import.tsv,
# to verbs.pl (the Alpino verb lexicon)
# The structure of alp_import.tsv is: verb <tab> conjugation <tab> complement

#####

BEGIN{
FS = "\t"   # Fields are separated by a tab
RS = "\n"   # Records are separated by a blank line

# Most verb tags in Alpino start with "v(" (the double slash is necessary, since
# the variable is used as regular expression)
verbtagn1 = "^v\\("

# Some verb tags in Alpino start with "verb_"
verbtagn2 = "verb_"

# Put verbs.pl in array alpino[y]
y=0
while(getline < "verbs.pl")
{
    alpino[y] = $0

    # Put the first line of each verbtagn in alpinoverb[y]
    if(alpino[y] ~ verbtagn1 || alpino[y] ~ verbtagn2) alpinoverb[y] = alpino[y]

    y++
}
}
```

```

{
# to make sure that the verb is not part of another verb, the verb is put in isolation
# with the reg expression [\[,]verb[\,]
verb = "\[\[\[, \]" $1 "\[\[\], \]"
complement = $3

conj = $2

# Alpino uses the label 'unacc' for unaccusative verbs.
# The mwes.tsv only specifies conjugation.
# To capture mismatches, conj can also be 'unacc' when it is 'zijn' in mwe.tsv.
# This represented using the the regular expression z([\uacc([
if(conj == "z") conj = conj "\(\([\[\|unacc\(\([\["
else conj = conj "\(\([\[" # add ([ to the conjugation

for (z=0; z<=y; z++)
{
# Look for a verb tag stored in alpinoverb that contains verb
if(alpinoverb[z] ~ verb)
{
# If the verb tag matches "v_root"
if(alpinoverb[z] ~ verbtage2)
{
# If the verb tag matches "verb_"
if(alpino[z+1] ~ verbtage3)
{
# Substitute the complement for the second line after the first line
# of the verb tag, followed by a comma, an enter and the original line.
# I.e. the complement is merged into the lexical entry just
# before the second complement listed in the verb tag.
alpino[z+3] = complement ",\n" alpino[z+3]
}
}
else # If the verb tag matches "v("
{
# Look within the verb tag
for (z=z+1; z<=y; z++)
{
# Look for a line that contains conj, which is either h([ or z([ or b([
if(alpino[z] ~ conj)
{
# Substitute the complement for the first line after the line with
# the conjugation, followed by a comma, an enter and the original line.
# I.e. the complement is merged into the lexical entry just before
# the second complement listed in the verb tag.
if(alpino[z+1] == "")
{
head_comp = substr(alpino[z],1,length(alpino[z])-5)

```


A.4 File structure of *mwes.tsv*

18 fields

- field 01** ID a unique identifier automatically assigned by the database
- field 02** PATTERN_NAME1 to specify the pattern name (see Grégoire (2007b) section 2.1 and 3.1.1)
- field 03** PATTERN_NAME2 to specify an additional pattern name (see Grégoire (2007b) section 2.1 and 3.1.1)
- field 04** PATTERN_NAME3 to specify a second additional pattern name (see Grégoire (2007b) section 2.1 and 3.1.1)
- field 05** EXPRESSION to represent the fixed components of the expression (see Grégoire (2007b) section 3.1.2)
- field 06** CL to represent the non-inflected forms of the fixed components followed by a parameter value (see Grégoire (2007b) section 3.1.3)
- field 07** LISTA to represent verbs (see Grégoire (2007b) section 3.1.4)
- field 08** LISTB to represent verbs or adjectives (see Grégoire (2007b) section 3.1.4)
- field 09** SUBJECT to represent predefined subject labels or heads of possible subjects (see Grégoire (2007b) section 3.2.1)
- field 10** OBJECT to represent predefined object labels or heads of possible objects (see Grégoire (2007b) section 3.2.2)
- field 11** RPRON to represent predefined R-pronominalization labels (see Grégoire (2007b) section 3.2.3)
- field 12** MODIFIER to represent possible modifiers (see Grégoire (2007b) section 3.2.4)
- field 13** EXAMPLE1 to represent an example sentence corresponding to PATTERN_NAME1 (see Grégoire (2007b) section 3.1.5)
- field 14** EXAMPLE2 to represent an example sentence corresponding to PATTERN_NAME2 (see Grégoire (2007b) section 3.1.5)
- field 15** EXAMPLE3 to represent an example sentence corresponding to PATTERN_NAME3 (see Grégoire (2007b) section 3.1.5)
- field 16** CONJUGATION to represent the conjugation of the head of the expression (see Grégoire (2007b) section 3.2.5)
- field 17** POLARITY to represent whether the expression occurs in a polarity environment (see Grégoire (2007b) section 3.2.6)
- field 18** COMMENTS to specify any comments

B Converted ECs

This appendix lists the most relevant details of the ECs converted to the Alpino lexicon. The following information is given: an example expression, the SP (source pattern), the TP (target pattern) and *alp_import* examples. More information can be found in in *patterns.tsv*, *mwes.tsv* and the related AWK script.

EC7

zijn debuut maken

SP = [.VP [.obj1:NP [.det:D (1)] [.hd:N1 (2)]] [.hd:V (3)]]

TP = fixed([acc(noun)],norm_passive)

```
stellen h part_fixed(in,[acc(commissie)],norm_passive)
krijgen h fixed([acc(opdoffer)],norm_passive)
krijgen h fixed([acc(opdonder)],norm_passive)
leveren h fixed([acc(topprestatie)],norm_passive)
halen h part_fixed(op,[acc(herinnering)],norm_passive)
```

EC8

risico lopen dat

SP = [.VP [.obj1:NP [.hd:N1 (1)]] [.hd:V (list)] [.vc:ssub (var)]]

TP = fixed([acc(noun),sbar],norm_passive)

```
houden h fixed([acc(hoop),sbar],norm_passive)
maken h fixed([acc(notitie),sbar],norm_passive)
stellen h fixed([acc(voorwaarde),sbar],norm_passive)
uiten h fixed([acc(kritiek),sbar],norm_passive)
leveren h fixed([acc(bewijs),sbar],norm_passive)
```

EC9

bod op

SP = [.VP [.obj1:NP [.hd:N1 (1)]] [.hd:V (list)] [.pc:PP [.hd:P (2)] [.obj1:NP (var)]]]

TP = fixed([acc(noun),pc(adposition)],norm_passive)

```
hebben h fixed([acc(bedenking),er_pp(bij)],norm_passive)
hebben h fixed([acc(bedenking),pc(bij)],norm_passive)
krijgen h fixed([acc(lol),er_pp(in)],norm_passive)
krijgen h fixed([acc(lol),er_pp(in,X),extra_vp(X)],norm_passive)
krijgen h fixed([acc(lol),pc(in)],norm_passive)
```

EC10

de moeite nemen om

SP = [.VP [.obj1:NP [.det:D (1)] [.hd:N (2)]] [.hd:V (3)] [.vc:OTI (var)]]

TP = fixed([[determiner,noun],vp],norm_passive)

tonen h fixed([[de,wil],vp],norm_passive)
geven h fixed([[bevel],vp],norm_passive)
krijgen h fixed([[mandaat],vp],norm_passive)
nemen h fixed([[de,vrijheid],vp],norm_passive)

EC11

iemand het gevoel geven dat

SP = [.VP [.obj2:NP (var)] [.obj1:NP [.det:D (1)] [.hd:N (2)]] [.hd:V (3)] [.vc:SSUB (var)]]

TP = fixed([[determiner,noun],dat,sbar],norm_passive)

geven h fixed([[het,gevoel],dat,sbar],norm_passive)
geven h fixed([[de,indruk],dat,sbar],norm_passive)
geven h fixed([[de,garantie],dat,sbar],norm_passive)

EC12 (commented pattern)

risico lopen dat

SP = [.VP [.hd:N1 (1)] [.hd:V (list)] [.vc:OTI (var)]]

TP = fixed([acc(noun),vp],norm_passive)

TPcount = fixed([[noun],vp],norm_passive)

COMMENTS = The nouns in this class behave both like count nouns (as they can both be plural and singular) and as mass nouns (as they can take an empty determiner and indefinites such as 'veel' and 'weinig').

ec12 examples

hebben h fixed([acc(behoefte),vp],norm_passive)
voelen h fixed([acc(behoefte),vp],norm_passive)
krijgen h fixed([acc(behoefte),vp],norm_passive)
hebben h fixed([acc(toestemming),vp],norm_passive)
krijgen h fixed([acc(toestemming),vp],norm_passive)

ec12count examples

hebben h fixed([[behoefte],vp],norm_passive)
voelen h fixed([[behoefte],vp],norm_passive)
krijgen h fixed([[behoefte],vp],norm_passive)
hebben h fixed([[toestemming],vp],norm_passive)
krijgen h fixed([[toestemming],vp],norm_passive)

EC13

alles geven

SP = [.VP [.obj1:PRON (1)] [.hd:V (2)]]

TP = fixed([acc(pronoun)],norm_passive)

geven h fixed([acc(alles)],norm_passive)
bijten h fixed([acc(elkaar)],norm_passive)
hebben h fixed([acc(iets)],norm_passive)
betekenen h fixed([acc(iets)],norm_passive)

EC14

twee vliegen in een klap slaan

SP = [.VP [.obj1:NP [.det:D (1)] [.hd:N (2)]] [.mod:PP [.hd:P (3)] [.obj1:NP [.det:D (4)] [.hd:N (5)]]] [.hd:V (6)]]
TP = fixed([[determiner1,noun1],[adposition,determiner2,noun2]],norm_passive)

krijgen h fixed([[een,klap],[in,het,gezicht]],norm_passive)
slaan h fixed([[twee,vliegen],[in,een,klap]],norm_passive)

EC16

de schuld krijgen van

SP = [.VP [.obj1:NP [.det:D (1)] [.hd:N (2)]] [.hd:V (3)] [.pc:PP [.hd:P (4)] [.obj1:NP (var)]]]]
TP = fixed([[determiner,noun],pc(adposition)],norm_passive) (A complement with the R-pronominalization representation *er_pp(adposition)* will be created automatically)

drijven h fixed([[de,spot],er_pp(met)],norm_passive)
drijven h fixed([[de,spot],pc(met)],norm_passive)
effenen h fixed([[het,pad],er_pp(voor)],norm_passive)
effenen h fixed([[het,pad],pc(voor)],norm_passive)
scheppen h fixed([[orde],er_pp(in)],norm_passive)
scheppen h fixed([[orde],pc(in)],norm_passive)

EC17

zich zorgen maken

SP = [.VP [.se:PRON (1)] [.predc:AP [.hd:N1 (2)]] [.hd:V (3)]]
TP = fixed([np_pred(noun),refl],norm_passive)

leggen h part_fixed(op,[np_pred(beperking),refl],norm_passive)
gunnen h fixed([np_pred(rust),refl],norm_passive)
doen h fixed([np_pred(pijn),refl],norm_passive)

EC19

zich in de schulden steken

SP = [.VP [.se:PRON (1)] [.ld:PP [.hd:P (2)] [.obj1:NP [.det:D (3)] [.hd:N (4)]]]] [.hd:V (5)]]
TP = fixed([[adposition,determiner,noun],refl],norm_passive)

steken h fixed([[in,de,schulden],refl],norm_passive)
snijden h fixed([[in,de,vingers],refl],norm_passive)
houden h fixed([[op,de,achtergrond],refl],norm_passive)

EC20

zich op zijn gemak voelen

SP = [.VP [.se:PRON (1)] [.predc:PP [.hd:P (2)] [.obj1:NP [.det:D (3)] [.hd:N (4)]]] [.hd:V (5)]]

TP = fixed([[adposition,determiner,noun],refl],norm_passive)

werken h fixed([[in,het,zweet],refl],norm_passive)
werken h fixed([[in,de,nesten],refl],norm_passive)
maken h fixed([[van,kant],refl],norm_passive)

EC21

zich druk maken

SP = [.VP [.se:PRON (1)] [.predc:AP [.hd:A1 (2)]] [.hd:V (3)]]

TP = fixed([ap_pred(adjective),refl],norm_passive)

maken h fixed([ap_pred(druk),refl],norm_passive)
rekenen h fixed([ap_pred(rijk),refl],norm_passive)

EC23

zich ergens vestigen

SP = [.VP [.ld (var)] [.hd:REFLV (2)]]

TP = refl_ld_pp (refl_ld.adv is created automatically)

spelen h part_refl_ld_adv(af)
spelen h part_refl_ld_pp(af)
scharen h refl_ld_adv
scharen h refl_ld_pp
breiden h part_refl_ld_adv(uit)
breiden h part_refl_ld_pp(uit)

EC24

iemand om het leven brengen

SP = [.VP [.obj1:NP (var)] [.predc:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N (3)]]] [.hd:V (list)]]

TP = fixed([[adposition,determiner,noun],acc],norm_passive)

brengen h fixed([[op,een,dwaalspoor],acc],norm_passive)
brengen h fixed([[in,cultuur],acc],norm_passive)
brengen h fixed([[in,trilling],acc],norm_passive)
brengen h fixed([[in,discrediet],acc],norm_passive)

EC26

de botte bijl hanteren

SP = [.VP [.obj1:NP [.det:D (1)] [.mod:A (2)] [.hd:N (3)]] [.hd:V (4)]]

TP = fixed([[determiner,adjective,noun]],norm_passive)

hanteren h fixed([[de,botte,bijl]],norm_passive)

nemen h fixed([[een,andere,loop]],norm_passive)

halen h fixed([[een,nat,pak]],norm_passive)

spinnen h fixed([[zuiver,garen]],norm_passive)

EC28

de zaak op de spits drijven

SP = [.VP [.obj1:NP [.det:D (1)] [.hd:N (2)]] [.ld:PP [.hd:P (3)] [.obj1:NP [.det:D (4)] [.hd:N (5)]]] [.hd:V (6)]]

TP = fixed([[determiner1,noun1],[adposition,determiner2,noun2]],norm_passive)

houden h fixed([[beide,benen],[op,de,grond]],norm_passive)

hebben h fixed([[een,dak],[boven,onze,hoofd]],norm_passive)

hebben h fixed([[een,dak],[boven,mijn,hoofd]],norm_passive)

hebben h fixed([[een,dak],[boven,je,hoofd]],norm_passive)

hebben h fixed([[een,dak],[boven,jouw,hoofd]],norm_passive)

hebben h fixed([[een,dak],[boven,haar,hoofd]],norm_passive)

hebben h fixed([[een,dak],[boven,jullie,hoofd]],norm_passive)

hebben h fixed([[een,dak],[boven,zijn,hoofd]],norm_passive)

hebben h fixed([[een,dak],[boven,uw,hoofd]],norm_passive)

alp_import examples

drijven h fixed([[de,zaak],[op,de,spits]],norm_passive)

nemen h fixed([[de,touwtjes],[in,handen]],norm_passive)

nemen h fixed([[het,heft],[in,handen]],norm_passive)

EC30

een verklaring hebben voor

SP = [.VP [.obj1:NP [.det:D (1)] [.hd:N1 (2)]] [.hd:V (3)] [.pc:PP [.hd:P (4)] [.obj1:NP (var)]]]]

TP = fixed([acc(noun),pc(adposition)],norm_passive)

geven h fixed([acc(oordeel),er_pp(over)],norm_passive)

geven h fixed([acc(oordeel),pc(over)],norm_passive)

maken h fixed([acc(overstap),er_pp(naar)],norm_passive)

maken h fixed([acc(overstap),pc(naar)],norm_passive)

EC31 (commented pattern)

kans maken op

SP = [.VP [.obj1:NP [.hd:N1 (1)] [.hd:V (list)] [.pc:PP [.hd:P (2)] [.obj1:NP (var)]]]

TP = fixed([acc(noun),pc(adposition)],norm_passive)

TPcount = fixed([[noun],pc(adposition)],norm_passive)

COMMENTS = The nouns in this class behave both like count nouns (as they can both be plural and singular) and as mass nouns (as they can take an empty determiner and indefinites such as 'veel' and 'weinig').

ec31 examples

hebben h fixed([acc(patent),er_pp(op)],norm_passive)

hebben h fixed([acc(patent),pc(op)],norm_passive)

geven h fixed([acc(toelichting),er_pp(op)],norm_passive)

geven h fixed([acc(toelichting),pc(op)],norm_passive)

ec31count examples

hebben h fixed([[patent],er_pp(op)],norm_passive)

hebben h fixed([[patent],pc(op)],norm_passive)

geven h fixed([[toelichting],er_pp(op)],norm_passive)

geven h fixed([[toelichting],pc(op)],norm_passive)

EC32

een verklaring hebben voor

SP = [.VP [.obj1:NP [.det:D (1)] [.hd:N1 (2)]] [.hd:V (3)] [.pc:PP [.hd:P (4)] [.obj1:NP (var)]]]

TP = fixed([acc(noun),pc(adposition)],norm_passive)

bewijzen h fixed([[lippendienst],dat],norm_passive)

knijpen h part_fixed(dicht, [[de,keel],dat],norm_passive)

geven h fixed([[kracht],dat],norm_passive)

EC37

een verklaring hebben voor

SP = [.VP [.obj1:NP [.det:D (1)] [.mod:A1 (2)] [.hd:N (3)]] [.hd:V (4)] [.pc:PP [.hd:P (5)] [.obj1:NP (var)]]]

TP = fixed([[determiner,adjective,noun],pc(adposition)],norm_passive)

hebben h fixed([[een,lage,dunk],er_pp(van)],norm_passive)

hebben h fixed([[een,lage,dunk],pc(van)],norm_passive)

trekken h fixed([[een,zware,wissel],er_pp(op)],norm_passive)

trekken h fixed([[een,zware,wissel],pc(op)],norm_passive)

EC40

op waarheid berusten

SP = [.VP [.pc:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N (3)]]] [.hd:V (4)]]

TP = fixed([[adposition,determiner,noun]],norm_passive)

komen z fixed([[tot,wasdom]],norm_passive)
berusten h fixed([[op,waarheid],sbar_subj],norm_passive)
berusten h fixed([[op,waarheid]],norm_passive)

EC41

iemand het gevoel geven te

SP = [.VP [.obj2:NP (var)] [.obj1:NP [.det:D (1)] [.hd:N (2)]] [.hd:V (3)] [.vc:TI (var)]]
TP = fixed([[determiner,noun],dat,vp],norm_passive)

geven h fixed([[het,gevoel],dat,vp],norm_passive)
geven h fixed([[de,indruk],dat,vp],norm_passive)

EC42

bevel geven om

SP = [.VP [.obj1:NP [.hd:N1 (1)]] [.hd:V (list)] [.vc:OTI (var)]]
TP = fixed([acc(noun),vp],norm_passive)

geven h fixed([acc(bevel),vp],norm_passive)
krijgen h fixed([acc(bevel),vp],norm_passive)
krijgen h fixed([acc(mandaat),vp],norm_passive)

EC43

aan boord gaan

SP = [.VP [.ld:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N (3)]]] [.hd:V (list)]]
TP = fixed([[adposition,determiner,noun]],norm_passive)

blijven z fixed([[aan,boord]],norm_passive)
gaan z fixed([[aan,boord]],norm_passive)
komen z fixed([[aan,de,deur]],norm_passive)
komen z fixed([[in,de,boeken]],norm_passive)

EC45

de neiging hebben om

SP = [.VP [.obj1:NP [.det:D (1)] [.hd:N2 (2)]] [.hd:V (3)] [.vc:OTI (var)]]
TP = fixed([acc(noun),vp],norm_passive)

hebben h fixed([acc(eer),vp],norm_passive)
smaken h fixed([acc(genoegen),vp],norm_passive)
hebben h fixed([acc(voorrecht),vp],norm_passive)

EC49

een verklaring hebben voor

SP = [.VP [.hd:REFLV (2)]]

TP = refl

vouwen h part_refl(op)

transformeren h refl

verzekeren h part_refl(bij)

EC50

het gevoel hebben te

SP = [.VP [.obj1:NP [.det:D (1)] [.hd:N2 (2)]] [.hd:V (3)] [.vc:TI (var)]]

TP = fixed([acc(noun),vp],norm_passive)

hebben h fixed([acc(idee),vp],norm_passive)

krijgen h fixed([acc(indruk),vp],norm_passive)

wekken h fixed([acc(suggestie),vp],norm_passive)

EC51

om het leven komen

SP = [.VP [.predc:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N (3)]]] [.hd:V (list)]]

TP = fixed([[adposition,determiner,noun]],norm_passive)

komen z fixed([[in,slaap]],norm_passive)

vallen z fixed([[in,slaap]],norm_passive)

gaan z fixed([[in,de,fout]],norm_passive)

zijn z fixed([[in,de,fout]],norm_passive)

EC52

iemand de laatste eer bewijzen

SP = [.VP [.obj2:NP (var)] [.obj1:NP [.det:D (1)] [.mod:A (2)] [.hd:N (3)]] [.hd:V (4)]]

TP = fixed([[determiner,adjective,noun],dat],norm_passive)

geven h fixed([[een,nieuwe,impuls],dat],norm_passive)

bewijzen h fixed([[de,laatste,eer],dat],norm_passive)

laten h fixed([[de,vrije,loop],dat],norm_passive)

EC54

de neiging hebben om

SP = [.VP [.se:PRON (1)] [.obj1:NP [.hd:N1 (2)]] [.hd:V (3)] [.pc:PP [.hd:P (4)] [.obj1:NP (var)]]]

TP = fixed([np_pred(noun),refl,pc(adposition)],norm_passive)

maken h fixed([np_pred(zorg), refl, er_pp(over)], norm_passive)
 maken h fixed([np_pred(zorg), refl, er_pp(over, X), extra_sbar(X)], norm_passive)
 maken h fixed([np_pred(zorg), refl, pc(over)], norm_passive)
 maken h fixed([np_pred(zorg), refl, er_pp(om)], norm_passive)
 maken h fixed([np_pred(zorg), refl, er_pp(om, X), extra_sbar(X)], norm_passive)
 maken h fixed([np_pred(zorg), refl, pc(om)], norm_passive)

EC55

iemand de weg wijzen

SP = [.VP [.obj2:NP (var)] [.obj1:NP [.det:D (1)] [.hd:N2 (2)]] [.hd:V (3)]]
 TP = fixed([acc(noun), dat], norm_passive)

geven h fixed([acc(schouderklop), dat], norm_passive)
 zeggen h part_fixed(aan, [acc(ontslag), dat], norm_passive)
 brengen h fixed([acc(bezoek), dat], norm_passive)
 bezorgen h fixed([acc(rilling), dat], norm_passive)

EC56

vraag stellen aan

SP = [.VP [.obj1:N1 [.hd:N (1)]] [.hd:V (list)] [.obj2:PP [.hd:P (aan)] [.obj1:NP (var)]]]
 fixed([acc(noun), dat_pp(aan)], norm_passive)

geven h fixed([acc(invulling), dat_pp(aan)], norm_passive)
 geven h fixed([acc(ruchtbaarheid), dat_pp(aan)], norm_passive)
 geven h fixed([acc(cachet), dat_pp(aan)], norm_passive)

EC57

iemand het recht geven om

SP = [.VP [.obj2:NP (var)] [.obj1:NP [.det:D (1)] [.hd:N (2)]] [.hd:V (3)] [.vc:OTI (var)]]
 TP = fixed([[determiner, noun], dat, vp], norm_passive)

geven h fixed([[de, gelegenheid], dat, vp], norm_passive)
 geven h fixed([[bevel], dat, vp], norm_passive)
 geven h fixed([[kracht], dat, vp], norm_passive)

EC58

iemand gezelschap houden

SP = [.VP [.obj1:NP (var)] [.predc:NP [.det:D (1)] [.hd:N (2)]] [.hd:V (3)]]
 TP = fixed([[determiner, noun], acc], norm_passive)

stellen h fixed([[kandidaat],acc],norm_passive)
houden h fixed([[gezelschap],acc],norm_passive)
maken h fixed([[een,compliment],acc],norm_passive)
maken h fixed([[het,hof],acc],norm_passive)

EC59

het woord geven aan

SP = [.VP [.obj1:NP [.det:D (1)] [.hd:N (2)]] [.hd:V (3)] [.obj2:PP [.hd:P (aan)] [.obj1:NP (var)]]]
TP = fixed([[determiner,noun],dat_pp(aan)],norm_passive)

geven h fixed([[uitdrukking],dat_pp(aan)],norm_passive)
bewijzen h fixed([[lippendienst],dat_pp(aan)],norm_passive)
geven h fixed([[college],dat_pp(aan)],norm_passive)
geven h fixed([[kracht],dat_pp(aan)],norm_passive)

EC60

iemand de gelegenheid geven tot

SP = [.VP [.obj2:NP (var)] [.obj1:NP [.det:D (1)] [.hd:N (2)]] [.hd:V (3)] [.pc:PP [.hd:P (4)] [.obj1:NP (var)]]]
TP = fixed([[determiner,noun],dat_pc(adposition)],norm_passive)

geven h fixed([[de,gelegenheid],dat_er_pp(tot)],norm_passive)
geven h fixed([[gelegenheid],dat_pc(tot)],norm_passive)
geven h fixed([[gelegenheid],dat_er_pp(tot)],norm_passive)
geven h fixed([[de,gelegenheid],dat_pc(tot)],norm_passive)

EC63

op vrije voeten komen

SP = [.VP [.predc:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.mod:A (3)] [.hd:N (4)]]] [.hd:V (list)]]
TP = fixed([[adposition,determiner,adjective,noun]],norm_passive)

komen z fixed([[op,vrije,voeten]],norm_passive)
zijn z fixed([[op,vrije,voeten]],norm_passive)
zijn z fixed([[van,tijdelijke,aard]],norm_passive)

EC65

iemand een aanbod doen

SP = [.VP [.obj2:NP (var)] [.obj1:NP [.det:D (1)] [.hd:N1 (2)]] [.hd:V (3)]]
TP = fixed([acc(noun),dat],norm_passive)

doen h fixed([acc(aanbod),dat],norm_passive)
leveren h fixed([acc(streek),dat],norm_passive)
geven h fixed([acc(applaus),dat],norm_passive)

EC67

zich aan de dood uitleveren

SP = [.VP [.se:PRON (1)] [.pc:PP [.hd:P (2)] [.obj1:NP [.det:D (3)] [.hd:N (4)]]] [.hd:V (5)]]

TP = fixed([[adposition,determiner,noun],refl],norm_passive)

leveren h part_fixed(uit,[[aan,de,dood],refl],norm_passive)
houden h fixed([[aan,de,regels],refl],norm_passive)
onthouden h fixed([[van,stemming],refl],norm_passive)

EC68

zich iets op de hals halen

SP = [.VP [.se:PRON (1)] [.obj1:NP (var)] [.ld:PP [.hd:P (2)] [.obj1:NP [.det:D (3)] [.hd:N (4)]]] [.hd:V (5)]]

TP = fixed([[adposition,determiner,noun],acc,refl],norm_passive)

halen h fixed([[op,de,hals],acc,refl],norm_passive)
houden h fixed([[van,het,lijf],acc,refl],norm_passive)
halen h fixed([[voor,de,geest],acc,refl],norm_passive)

EC69

krab bij kas zitten

SP = [.VP [.predc:AP [.hd:A1 (1)] [.mod:PP [.hd:P (2)] [.obj1:NP [.det:D (3)] [.hd:N (4)]]]] [.hd:V (5)]]

TP = fixed([[adjective,adposition,determiner,noun]],norm_passive)

lopen z fixed([[hard,van,stapel]],norm_passive)
staan h fixed([[hoog,in,aanzien]],norm_passive)
zitten h fixed([[krap,bij,kas]],norm_passive)

EC71

de koppen bij elkaar steken

SP = [.VP [.obj1:NP [.det:D (1)] [.hd:N (2)]] [.ld:PP [.hd:P (3)] [.obj1:NP [.hd:PRON (4)]]] [.hd:V (5)]]

TP = fixed([[determiner,noun],[adposition,pronoun]],norm_passive)

knopen h fixed([[de,eindjes],[aan,elkaar]],norm_passive)
steken h fixed([[de,koppen],[bij,elkaar]],norm_passive)
laden h fixed([[de,verdenking],[op,zich]],norm_passive)

EC73

iemand tegen de borst stuiten

SP = [.VP [.obj2:NP (var)] [.ld:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N (3)]]] [.hd:V (4)]]

TP = fixed([[adposition,determiner,noun],dat],norm_passive)

stuiten h fixed([[tegen,de,borst],dat,sbar_subj],norm_passive)

stuiten h fixed([[tegen,de,borst],dat,sbar_subj_no_het],norm_passive)

stuiten h fixed([[tegen,de,borst],dat],norm_passive)

EC74

iemand aan de haak slaan

SP = [.VP [.obj1:NP (var)] [.ld:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N (3)]]] [.hd:V (4)]]

TP = fixed([[adposition,determiner,noun],acc],norm_passive)

zetten h fixed([[op,de,rails],acc],norm_passive)

leiden h fixed([[om,de,tuin],acc],norm_passive)

slaan h fixed([[aan,de,haak],acc],norm_passive)

EC77

iemand ten goede komen

SP = [.VP [.obj2:NP (var)] [.ld:PP fixed(1 2)] [.hd:V (list)]]

TP = fixed([[x1,x2],dat],norm_passive)

komen z fixed([[ten,goede],dat],norm_passive)

vallen z fixed([[ten,deel],dat],norm_passive)

schieten z fixed([[te,hulp],dat],norm_passive)

komen z fixed([[te,hulp],dat],norm_passive)

EC78

iemand ten goede komen

SP = [.VP [.obj1:NP (var)] [.ld:PP fixed(1 2)] [.hd:V (list)]]

TP = fixed([[x1,x2],acc],norm_passive)

brengen h fixed([[ten,uitvoer],acc],norm_passive)

spreiden h fixed([[ten,toon],acc],norm_passive)

leggen h fixed([[te,vondeling],acc],norm_passive)

staan h fixed([[ter,zijde],acc],norm_passive)

EC79

in de la liggen

SP = [.VP [.ld:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N (3)]]] [.hd:V (4)]]
TP = fixed([[adposition,determiner,noun]],norm_passive)

leven h fixed([[in,angst]],norm_passive)
zitten h fixed([[in,angst]],norm_passive)
liggen h fixed([[in,de,la]],norm_passive)

EC80

iemand het mes op de keel zetten

SP = [.VP [.obj2:NP (var)] [.obj1:NP [.det:D (1)] [.hd:N (2)]]] [.ld:PP [.hd:P (3)] [.obj1:NP [.det:D (4)] [.hd:N (5)]]] [.hd:V (6)]]
TP = fixed([[determiner1,noun1],[adposition,determiner2,noun2],dat],norm_passive)

zetten h fixed([[het,mes],[op,de,keel],dat],norm_passive)
geven h fixed([[een,duwtje],[in,de,rug],dat],norm_passive)
nemen h fixed([[de,wind],[uit,de,zeilen],dat],norm_passive)

EC81

iets op een laag pitje zetten

SP = [.VP [.obj1:NP (var)] [.ld:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.mod:A (3)] [.hd:N (4)]]] [.hd:V (5)]]
TP = fixed([[adposition,determiner,adjective,noun],acc],norm_passive)

zetten h fixed([[op,een,laag,pitje],acc],norm_passive)
zetten h fixed([[op,losse,schroeven],acc],norm_passive)
gooien h fixed([[over,een,andere,boeg],acc],norm_passive)
brengen h fixed([[op,het,verkeerde,idee],acc],norm_passive)

EC82

met de gebakken peren zitten

SP = [.VP [.mod:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.mod:A (3)] [.hd:N (4)]]] [.hd:V (5)]]
TP = fixed([[adposition,determiner,adjective,noun]],norm_passive)

zitten h fixed([[met,de,gebakken,peren]],norm_passive)
zitten h fixed([[met,kromme,tenen]],norm_passive)

EC83

de vinger op de zere plek leggen

SP = [.VP [.obj1:NP [.det:D (1)] [.hd:N (2)]] [.ld:PP [.hd:P (3)] [.obj1:NP [.det:D (4)] [.mod:A (5)] [.hd:N (6)]]] [.hd:V (7)]]

TP = fixed([[determiner1,noun1],[adposition,determiner2,adjective,noun2]],norm_passive)

hebben h fixed([[het,heft],[in,eigen,handen]],norm_passive)

leggen h fixed([[de,vinger],[op,de,zere,plek]],norm_passive)

nemen h fixed([[het,recht],[in,eigen,hand]],norm_passive)

EC84

met de handen in het haar zitten

SP = [.VP [.mod:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N (3)]]] [.ld:PP [.hd:P (4)] [.obj1:NP [.det:D (5)] [.hd:N (6)]]] [.hd:V (7)]]

TP = fixed([[adposition1,determiner1,noun1],[adposition2,determiner2,noun2]],norm_passive)

zitten h fixed([[met,de,handen],[in,het,haar]],norm_passive)

staan h fixed([[met,onze,rug],[tegen,de,muur]],norm_passive)

staan h fixed([[met,mijn,rug],[tegen,de,muur]],norm_passive)

staan h fixed([[met,je,rug],[tegen,de,muur]],norm_passive)

staan h fixed([[met,jouw,rug],[tegen,de,muur]],norm_passive)

staan h fixed([[met,haar,rug],[tegen,de,muur]],norm_passive)

staan h fixed([[met,jullie,rug],[tegen,de,muur]],norm_passive)

staan h fixed([[met,zijn,rug],[tegen,de,muur]],norm_passive)

staan h fixed([[met,uw,rug],[tegen,de,muur]],norm_passive)

staan h fixed([[met,de,rug],[tegen,de,muur]],norm_passive)

EC87

in aanmerking komen om

SP = [.VP [.predc:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N (3)]]] [.hd:V (list)] [.vc:OTI (var)]]

TP = fixed([[adposition,determiner,noun],vp],norm_passive)

komen z fixed([[in,de,gelegenheid],vp],norm_passive)

komen z fixed([[in,de,verleiding],vp],norm_passive)

EC88

in aanmerking komen voor

SP = [.VP [.predc:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N (3)]]] [.hd:V (list)] [.pc:PP [.hd:P (4)] [.obj1:NP (var)]]]]

TP = fixed([[adposition1,determiner,noun],pc(adposition2)],norm_passive)

komen z fixed([[in,aanmerking],er_pp(voor)],norm_passive)
komen z fixed([[in,aanmerking],er_pp(voor,X),extra_vp(X)],norm_passive)
komen z fixed([[in,aanmerking],pc(voor)],norm_passive)

EC90

tot de conclusie komen dat

SP = [.VP [.predc:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N (3)]]] [.hd:V (list)] [.vc:SSUB (var)]]

TP = fixed([[adposition,determiner,noun],sbar],norm_passive)

blijken z fixed([[van,mening],sbar],norm_passive)
blijven z fixed([[van,mening],sbar],norm_passive)
komen z fixed([[tot,de,conclusie],sbar],norm_passive)

EC92

stevig in het zadel zitten

SP = [.VP [.ld:PP [.mod:A1 (1)] [.hd:P (2)] [.obj1:NP [.det:D (3)] [.hd:N (4)]]] [.hd:V (5)]]

TP = fixed([[adjective,adposition,determiner,noun]],norm_passive)

zitten h fixed([[stevig,in,het,zadel]],norm_passive)
liggen h fixed([[vers,in,het,geheugen],sbar_subj],norm_passive)
liggen h fixed([[vers,in,het,geheugen],sbar_subj_no_het],norm_passive)
liggen h fixed([[vers,in,het,geheugen]],norm_passive)
zitten h fixed([[strak,in,het,pak]],norm_passive)

EC93

iemand iets in handen geven

SP = [.VP [.obj2:NP (var)] [.obj1:NP (var)] [.ld:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N (3)]]] [.hd:V (4)]]

TP = fixed([[adposition,determiner,noun],acc,dat],norm_passive)

EC94

ten onder gaan

SP = [.VP [.ld:PP fixed(1 2)] [.hd:V (list)]]

TP = fixed([[x1,x2]],norm_passive)

EC97

met vuur spelen

SP = [.VP [.mod:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N (3)]]] [.hd:V (4)]]
TP = fixed([[adposition,determiner,noun]],norm_passive)

EC98

voor hetere vuren staan

SP = [.VP [.ld:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.mod:A (3)] [.hd:N (4)]]] [.hd:V (5)]]
TP = fixed([[adposition,determiner,adjective,noun]],norm_passive)

EC99

uit elkaar gaan

SP = [.VP [.ld:PP [.hd:P (1)] [.obj1:NP [.hd:PRON (2)]]] [.hd:V (3)]]
TP = fixed([[adposition,pronoun]],norm_passive)

EC100

iemand in elkaar slaan

SP = [.VP [.obj1:NP (var)] [.ld:PP [.hd:P (1)] [.obj1:NP [.hd:PRON (2)]]] [.hd:V (3)]]
TP = fixed([[adposition,pronoun],acc],norm_passive)

EC102

iemand in aanraking brengen met

SP = [.VP [.obj1:NP (var)] [.predc:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N (3)]]] [.hd:V (list)] [.pc:PP [.hd:P (3)] [.obj1:NP (var)]]]
TP = fixed([[adposition1,determiner,noun],acc,pc(adposition2)],norm_passive)

EC105

ten grondslag liggen aan

SP = [.VP [.ld:PP fixed(1 2)] [.hd:V (list)] [.pc:PP [.hd:P (3)] [.obj1:NP (var)]]]

TP = fixed([[x1,x2],pc(adposition)],norm_passive)

EC106

iemand op zijn woord geloven

SP = [.VP [.obj1:NP (var)] [.mod:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N (3)]]]] [.hd:V (4)]]

TP = fixed([[adposition,determiner,noun],acc],norm_passive)

EC109

in de rij staan voor

SP = [.VP [.ld:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N (3)]]]] [.hd:V (4)] [.pc:PP [.hd:P (5)] [.obj1:NP (var)]]]]

TP = fixed([[adposition2,determiner,noun],pc(adposition2)],norm_passive)

EC110

iemand te woord staan

SP = [.VP [.obj1:NP (var)] [.predc:PP fixed(1 2)] [.hd:V (list)]]

TP = fixed([[x1,x2],acc],norm_passive)

EC111

liefde op het eerste gezicht zijn

SP = [.VP [.obj1:NP [.det:D (1)] [.hd:N (2)] [.mod:PP [.hd:P (3)] [.obj1:NP [.det:D (4)] [.mod:A (5)] [.hd:N (6)]]]]]] [.hd:V (list)]]

TP = fixed([[determiner1,noun1,adposition,determiner2,adjective,noun2]],norm_passive)

EC112

iets hoog in het vaandel hebben

SP = [.VP [.obj1:NP (var)] [.ld:PP [.mod:A1 (1)] [.hd:P (2)] [.obj1:NP [.det:D (3)] [.hd:N (4)]]] [.hd:V (5)]]

TP = fixed([[adjective,adposition,determiner,noun],acc],norm_passive)

EC114

aan het langste eind trekken

SP = [.VP [.pc:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.mod:A (3)] [.hd:N (4)]]] [.hd:V (5)]]

TP = fixed([[adposition,determiner,adjective,noun]],norm_passive)

EC115

de indruk krijgen dat

SP = [.VP [.obj1:NP [.det:D (1)] [.hd:N2 (2)]]] [.hd:V (3)] [.vc:SSUB (var)]]

TP = fixed([acc(noun),sbar],norm_passive)

EC116

iemand welkom heten

SP = [.VP [.obj1:NP (var)] [.predc:AP [.hd:A1 (1)]]] [.hd:V (2)]]

TP = fixed([ap_pred(adjective),acc],norm_passive)

EC117

de kat de bel aanbinden

SP = [.VP [.obj2:NP [.det:D (1)] [.hd:N (2)]]] [.obj1:NP [.det:D (3)] [.hd:N (4)]]] [.hd:V (5)]]

TP = fixed([[determiner1,noun1],[determiner2,noun2]],norm_passive)

EC118

iemand een hak zetten

SP = [.VP [.obj2:NP (var)] [.predc:NP [.det:D (1)] [.hd:N (2)]] [.hd:V (3)]]

TP = fixed([[determiner,noun],dat],norm_passive)

EC120

het recht verwerven op

SP = [.VP [.obj1:NP [.det:D (1)] [.hd:N2 (2)]] [.hd:V (3)] [.pc:PP [.hd:P (4)] [.obj1:NP (var)]]]]

TP = fixed([acc(noun),pc(adposition)],norm_passive)

EC126

water in de wijn doen

SP = [.VP [.obj1:NP [.hd:N2 (1)]] [.ld:PP [.hd:P (2)] [.obj1:NP [.det:D (3)] [.hd:N (4)]]] [.hd:V (5)]]

TP = fixed([acc(noun),[adposition,determiner,noun]],norm_passive)

EC127

de macht over het stuur verliezen

SP = [.VP [.obj1:NP [.det:D (1)] [.hd:N (2)] [.mod:PP [.hd:P (3)] [.obj1:NP [.det:D (4)] [.hd:N (5)]]]] [.hd:V (6)]]

TP = fixed([[determiner1,noun1,adposition,determiner2,noun2]],norm_passive)

EC131

ten goede komen aan

SP = [.VP [.ld:PP fixed(1 2)] [.hd:V (list)] [.obj2:PP [.hd:P (aan)] [.obj1:NP (var)]]]]

TP = fixed([[x1,x2],dat_pp(aan)],norm_passive)

EC133

knoop doorhakken

SP = [.VP [.obj1:NP [.hd:N1 (1)]] [.hd:V (2)]]

TP = fixed([acc(noun)],norm_passive)

EC134 (commented pattern)

iemand raad geven

SP = [.VP [.obj2:NP (var)] [.obj1:N1 [.hd:N (1)]] [.hd:V (list)]]

TP = fixed([acc(noun),dat],norm_passive)

TPcount = fixed([[noun],dat],norm_passive)

COMMENTS = The nouns in this class behave both like count nouns (as they can both be plural and singular) and as mass nouns (as they can take an empty determiner and indefinites such as 'veel' and 'weinig').

ec134 examples

ec134count examples

EC135 (commented pattern)

ihoud geven aan iets

SP = [.VP [.obj1:N1 [.hd:N (1)]] [.hd:V (list)] [.obj2:NP [.hd:P (aan)] [.obj1:NP (var)]]]

TP = fixed([acc(noun),dat_pp(aan)],norm_passive)

TPcount = fixed([[noun],dat_pp(aan)],norm_passive)

COMMENTS = The nouns in this class behave both like count nouns (as they can both be plural and singular) and as mass nouns (as they can take an empty determiner and indefinites such as 'veel' and 'weinig').

ec135 examples

ec135count examples

EC136

een bezoek brengen aan

SP = [.VP [.obj1:NP [.det:D (1)] [.hd:N2 (2)]] [.hd:V (3)] [.obj2:PP [.hd:P (aan)] [.obj1:NP (var)]]]

TP = fixed([acc(noun),dat_pp(aan)],norm_passive)

EC137

iemand opdracht geven om

SP = [.VP [.obj2:NP (var)] [.obj1:NP [.hd:N1 (1)] [.hd:V (list)] [.vc:OTI (var)]]

TP = fixed([acc(noun),dat,vp],norm_passive)

EC138

iemand iets duidelijk maken

SP = [.VP [.obj2:NP (var)] [.obj1:NP (var)] [.predc:AP [.hd:A1 (1)] [.hd:V (2)]]

TP = fixed([ap_pred(adjective),acc,dat],norm_passive)

EC139

iemand de belofte doen dat

SP = [.VP [.obj2:NP (var)] [.obj1:NP [.hd:N1 (1)] [.hd:V (list)] [.vc:SSUB (var)]]

TP = fixed([acc(noun),dat,sbar],norm_passive)

EC141

de slappe lach hebben

SP = [.VP [.obj1:NP [.det:D (1)] [.mod:A (2)] [.hd:N (3)]] [.hd:V (list)]]

TP = fixed([[determiner,adjective,noun]],norm_passive)

C Excluded ECs

This appendix lists the ECs that were excluded from the conversion. The source pattern (SP) and the reason for its exclusion from the conversion are given.

EC15

SP: [.VP [.obj1:NP [.hd:N1 (1)]] [.hd:V (2)] [.ld:PP [.hd:P (3)] [.obj1:NP (var)]]]
Reason: Unknown Alpino representation for a locative/directional PP with variable complement

EC18

SP: [.VP [.hd:REFLV (2)] [.pc:PP [.hd:P (3)] [.obj1:NP (var)]]]
Reason: Unknown Alpino pattern

EC22

SP: [.VP [.hd:REFLV (2)] [.ld:PP [.hd:P (3)] [.obj1:NP (var)]]]
Reason: Unknown Alpino representation for a locative/directional PP with variable complement

EC25

SP: [.VP [.ld:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N1 (3)]]] [.hd:V (list)]]
Reason: Unknown Alpino representation for a modifiable noun within a PP

EC27

SP: [.VP [.obj1:NP [.det:D (1)] [.mod:AP (var)] [.hd:N1 (2)]]] [.hd:V (3)]]
Reason: Unknown Alpino representation for an obligatory variable modifier

EC29

SP: [.VP [.su:NP [.det:D (1)] [.hd:N (2)]]] [.hd:V (3)]]
Reason: Unknown Alpino pattern

EC33

SP: [.VP [.obj1:NP [.det:D (1)] [.hd:N (2)]]] [.hd:V (3)] [.vc:INF [.su:NP [.det:D (1)] [.hd:N (2)]]] [.hd:V (4)]]]

Expression: zijn hoofd laten hangen

TP: fixed([vc(hang,inf,intransitive),[zijn hoofd]],no-passive), Reason: Unsure about Alpino pattern

EC34

SP: [.VP [.obj1:NP [.det:D (1)] [.hd:N (2)]]] [.hd:V (3)] [.ld:PP [.hd:P (4)] [.obj1:NP (var)]]]
Reason: Unknown Alpino representation for locative/directional PP with variable complement

EC35

SP: [.VP [.obj1:NP [.det:D (1)] [.hd:N (2)]] [.hd:V (3)] [.mod:PP [.hd:P (4)] [.obj1:NP (var)]]]

Reason: Unknown Alpino representation for variable modifier PP

EC36

SP: [.VP [.obj1:NP [.mod:A1 (1)] [.hd:N (2)]] [.hd:V (list)]]

Reason: Unknown how to represent an NP with fixed modifier and fixed noun, but variable determiner in Alpino

EC38

SP: [.VP [.obj2:NP (var)] [.obj1:NP [.det:D (1)] [.mod:AP (var)] [.hd:N1 (2)]] [.hd:V (3)]]

Reason: Unknown Alpino representation for an obligatory variable modifier

EC39

SP: [.VP [.obj1:NP [.det:D (1)] [.mod:AP (var)] [.hd:N1 (2)]] [.hd:V (3)] [.obj2:PP [.hd:P (aan)] [.obj1:NP (var)]]]

Reason: Unknown Alpino representation for an obligatory variable modifier

EC44

SP: [.VP [.obj1:NP [.det:D (1)] [.hd:N2 (2)]] [.hd:V (3)] [.mod:PP [.hd:P (4)] [.obj1:NP (var)]]]

Reason: Unknown Alpino representation for a PP modifier with variable complement

EC46

SP: [.PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N (3)]]]

Reason: non-verbal MWE

EC47

SP: [.VP [.obj1:NP (var)] [.ld:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N2 (3)]]] [.hd:V (4)]]

Reason: Unknown Alpino representation for a limitedly modifiable noun within a PP

EC48

SP: [.mod:PP [.hd:P fixed(1 2 3)] [.obj1:NP (var)]]

Reason: non-verbal MWE

EC53

SP: [.VP [.ld:PP [.mod:AdvP (var)] [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N (3)]]] [.hd:V (4)]]

Reason: Unknown Alpino representation for an obligatory variable modifier in the locative/directional complement

EC61

SP: [.VP [.obj1:NP [.det:D (1)] [.hd:N1 (2)]] [.hd:V (3)] [.ld:PP [.hd:P (4)] [.obj1:NP (var)]]]]

Reason: Unknown Alpino representation for a locative/directional PP with variable complement

EC62

SP: [.VP [.ld:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N2 (3)]]]] [.hd:V (4)]]

Reason: Unknown Alpino representation for a limitedly modifiable noun within a PP

EC64

SP: [.VP [.obj1:NP [.hd:N1 (1)] [.mod:PP [.hd:P (2)] [.obj1:NP (var)]]]] [.hd:V (list)]]

Reason: Unknown Alpino representation for a PP modifier with variable complement

EC66

SP: [.mod:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N (3)] [.mod:PP [.hd:P (4)] [.obj1:NP (var)]]]]]]

Reason: non-verbal MWE

EC70

Reason: unanalyzed EC, mostly unique patterns

EC72

SP: [.VP [.ld:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.mod:AP (var)] [.hd:N1 (3)]]]] [.hd:V (4)]]

Reason: Unknown Alpino representation for a variable modifier PP

EC75

SP: [.VP [.obj1:NP (var)] [.ld:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.mod:AP (var)] [.hd:N1 (3)]]]] [.hd:V (4)]]

Reason: Unknown Alpino representation for a variable modifier PP

EC76

SP: [.NP [.mod:A1 (list)] [.hd:N1 (1)]]

Reason: non-verbal MWE

EC85

SP: [.NP [.det:D (1)] [.mod:A (2)] [.hd:N (3)]]

Reason: non-verbal MWE

EC86

SP: [.VP [.predc:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N1 (3)]]] [.hd:V (list)]]

Reason: Unknown Alpino representation for a modifiable noun within a PP

EC89

SP: [.VP [.obj1:NP (var)] [.predc:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N2 (3)]]] [.hd:V (list)]]

Reason: Unknown Alpino representation for a limitedly modifiable noun within a PP

EC91

SP: [.VP [.obj1:NP (var)] [.predc:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N1 (3)]]] [.hd:V (list)]]

Reason: Unknown Alpino representation for a modifiable noun within a PP

EC95

SP: [.VP [.pc:PP [.hd:P (1)] [.obj1:NP [.hd:N1 (2)]]] [.hd:V (3)]]

Reason: Unknown Alpino representation for a modifiable noun within a PP

EC96

SP: [.VP [.ld:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N1 (3)]]] [.hd:V (4)]]

Reason: Unknown Alpino representation for a modifiable noun within a PP

EC101

SP: [.NP [.mod:A (1)] [.hd:N (2)]]

Reason: non-verbal MWE

EC103

SP: [.VP [.ld:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N1 (3)] [.mod:PP [.hd:P (4)] [.obj1:NP (var)]]]] [.hd:V (list)]]

Reason: Unknown Alpino representation for a PP modifier with variable complement

EC104

SP: [.VP [.ld:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N1 (3)]]] [.hd:V (list)] [.pc:PP [.hd:P (4)] [.obj1:NP (var)]]]]

Reason: Unknown Alpino representation for a modifiable noun within a PP

EC107

SP: [.VP [.obj1:NP (var)] [.ld:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N1 (3)]]] [.hd:V (4)]]

Reason: Unknown Alpino representation for a modifiable noun within a PP

EC108

SP: [.VP [.obj1:NP (var)] [.predc:AdvP [.hd:Adv (1)]] [.hd:V (1)]]

Reason: Unknown Alpino representation for a predicative adverb

EC113

SP: [.NP [.mod:A1 (1)] [.hd:N (2)]]

Reason: non-verbal MWE

EC119

SP: [.PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.mod:A (3)] [.hd:N (4)]]]]

Reason: non-verbal MWE

EC121

SP: [.VP [.predc:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N2 (3)]]]] [.hd:V (list)]]

Reason: Unknown Alpino representation for a limitedly modifiable noun within a PP

EC122

SP: [.NP [.det:D (1)] [.mod:A (2)] [.hd:N (3)] [.mod:PP [.hd:P (4)] [.obj1:NP [.det:D (5)] [.hd:N (6)]]]]]]

Reason: non-verbal MWE

EC123

SP: [.NP [.det:D (1)] [.hd:N (2)] [.mod:PP [.hd:P (3)] [.obj1:NP [.det:D (4)] [.hd:N (5)]]]]]]

Reason: non-verbal MWE

EC124

SP: [.VP [.obj1:NP [.det:D (1)] [.hd:N (2)] [.mod:PP [.hd:P (3)] [.obj1:NP (var)]]]] [.hd:V (4)]]

Reason: Unknown Alpino representation for a PP modifier with variable complement

EC125

SP: [.NP [.hd:N (1)] [.mod:PP [.hd:P (2)] [.obj1:NP [.det:D (3)] [.hd:N (4)]]]]]]

Reason: non-verbal MWE

EC128

SP: [.PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N (3)] [.mod:PP [.hd:P (4)] [.obj1:NP [.det:D (5)] [.hd:N (6)]]]]]

Reason: non-verbal MWE

EC129

SP: [.AP [.hd:A (1)] [.mod:AdvP [.hd:Adv (2)] [.obcomp:CP [.cmp:C (3)] [.body:NP [.det:D (4)] [.hd:N (5)]]]]]

Reason: non-verbal MWE

EC130

Not existing

EC132

SP: [.VP [.ld:PP [.hd:P (1)] [.obj1:NP [.det:D (2)] [.hd:N (3)] [.mod:PP [.hd:P (4)] [.obj1:NP (var)]]]] [.hd:V (list)]]

Reason: Unknown Alpino representation for a PP modifier with variable complement

EC140

SP: [.VP [.NP [.mod:A (1)] [.hd:N (2)]]] [.hd:V (list)]]

Reason: Unknown how to represent an NP with fixed adjective and variable determiner in Alpino