Paraphrasing factoid dependency trees into fluent sentences in a natural language

Author
Louis Onrust (s0723738)
l.onrust@student.science.ru.nl

Supervised by
Cornelis H. A. Koster
kees@cs.ru.nl

A feasibility study

Monday 21 June, 2010

Abstract

In this thesis the results are presented of a feasibility study of paraphrasing factoid dependency trees. It covers the analysis of a factoid, a sentence in which only elements of factual meaning are expressed, into a dependency graph, an abstract syntax for the factoid. The syntactic and semantic aspects that are preserved and the aspects that are eliminated are discussed. When paraphrasing from a dependency graph to a factoid, one has to compensate for these missing aspects. This thesis investigates what aspects are involved and what the consequences are.

The results indicated that simple paraphrasis of factoid dependency trees if the dependency trees are based on the aboutness-based model, is possible and shows great potential.

Contents

1 Introduction

This thesis concludes the 9 ECTS computer science bachelor's thesis course at the Radboud University Nijmegen.

1.1 Problem definition

In this thesis I will present the results of a feasibility study of paraphrasing factoid dependency trees into fluent sentences in a natural language. We do not address the paraphrasis of full text, with anaphora and discourse structures, but limit ourselves to factual sentences (factoids). We do not address the aspects of emotion, modality, rethorics, irony, poetry and style. We also do not address the conservation of meaning in paraphrasis, but the conservation of aboutness. We base ourselves on the DUPIRA parser, which translates Dutch sentences into dependency trees following an aboutness-based dependency model. Based on those results we will answer the following research questions:

1. How can a factoid sentence be analysed into an aboutness-based dependency graph?

2. What syntactic and semantic aspects of the original sentence are lost in an aboutness-based dependency analysis?

3. How can the dependency tree deriving from a certain factoid be paraphrased into another factoid with the same aboutness?

4. How can it be achieved that this paraphrase leads to a fluent and natural looking sentence?

1.2 Significance

In processing natural language, the internal representation of sentences is stored in an abstract form, e.g., as a dependency tree. In this conversion, certain information gets lost, such as word order. Therefore it is not always possible to restore the original from the dependency tree. The best one can expect is to arrive at another sentence with the same aboutness (paraphrasis). The goal of this thesis is to investigate the paraphrasis of dependency trees to fluent and natural looking sentences.

1.3 Background

Paraphrasing is the subject of research in many disciplines: linguistics, artificial intelligence (AI), computing science, &c. Their interests are not always the same. The linguists probably want to produce the most realistic, the most semantically correct paraphrase; whereas someone from artificial intelligence might be interested in the way paraphrasing can be learned by a machine. Therefore there is a discrepancy between the results of the researches to paraphrasing. AI research into paraphrasing is mostly based on machine learning. This group of researchers appears to be very close, and hardly any publication is made publicly available. This is the reason why there is not much (relevant) information to be found on this subject. Therefore we will shortly introduce the concepts that are critical to this thesis.

1.3.1 What is a dependency tree?

Dependency trees (actually directed acyclic *dependency graphs*) consist recursively of dependency triples. A *dependency triple* is defined as [word, relation, word]. In this thesis, as in most literature, the terms dependency graph and dependency tree will be interchangeable.

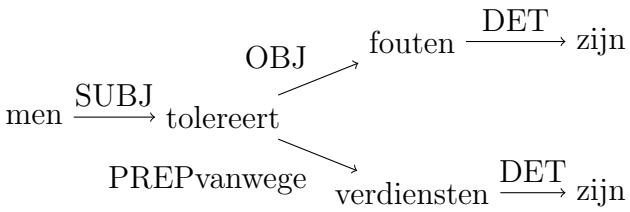
The definition of a dependency graph in [11] can be adapted to our dependency graphs:

- Let $R = r_1, \ldots, r_m$ be a set of relations, the dependency types;
- A dependency graph for a string of words $W = w_1 \ldots w_n$ is a labelled directed graph $G = (W, A)$, where
  - W is the set of nodes, i.e. word tokens in the input string;
  - A is a set of labelled arcs $[w_i, r, w_j]$; $w_i, w_j \in W$, $r \in R$.
- A dependency graph G is well-formed iff it is acyclic.

A major difference with the definition in [11] is that we allow nodes to have an in-degree greater than 1. This means we see a dependency graph as a dependency tree with possible additional in-going arcs.

If a sentence is transduced to dependency triples, the syntactic relations between the words are known. Thus the sentence is represented in an abstract way. We will use the notation $w_i \xrightarrow{r} w_j$ to show $[w_i, r, w_j] \in A$. The convention is to call w_i the head, and w_j the modifier.

This example of a well-formed dependency tree is obtained by transducing the Dutch sentence; "Men tolereert zijn fouten vanwege zijn verdiensten":



This tree can be decomposed into the triples:

[men, SUBJ, tolereert]
[tolereert, OBJ, fouten]
[fouten, DET, zijn]
[tolereert, PREPvanwege, verdiensten]
[verdiensten, DET, zijn]

The transduction from sentences to such tree is done by DUPIRA, the Dutch Parser for Information Retrieval Applications. This parser finds the most plausible syntactic analysis of a Dutch sentence and transduces it to one or more dependency graphs.

1.3.2 What is a factoid?

A *factoid* is a sentence in which only elements of factual meaning are expressed. A factoid is typically of the form: subject – verb – object – complements. For example: "Who did what to whom and where".

1.3.3 What is paraphrasis?

Paraphrasing is the act of reformulating a text or sentence with the intention to clarify or to explain. Reformulating in such a manner can be done in several ways, to mention a few: lexical paraphrasis, semantic paraphrasis and syntactic paraphrasis. In this thesis the focus will be on the last one: syntactic paraphrasis.

In [5] *paraphrasis* is defined in three definitions:

- A paraphrase pair is a pair of units of text considered as interchangeable;
- A paraphrase mapping is the mapping between the two elements of a paraphrase pair;
- A paraphrase of a unit of text is the alternative given to that unit of text in some paraphrase pair.

Syntactic paraphrasis In the context of this thesis, paraphrasing is a way to transform dependency graphs into natural looking, fluent sentences. The dependency graphs consist of triples containing syntactic relations, therefore it is natural to do syntactic paraphrasis.

In an ideal situation, paraphrasing a dependency graph will yield the original sentence on which the dependency graph was based. It is easy to see that this is not always possible, given the dependency graph from §1.3.1. If we paraphrase it, there are at least two possible outcomes:

- Men tolereert vanwege zijn verdiensten zijn fouten
- Men tolereert zijn fouten vanwege zijn verdiensten

Although these sentences look alike, they might represent another semantic meaning: *tolereren* puts a lot of stress on the succeeding XP. The first sentence stresses the merits (*verdiensten*), as they are more important than its mistakes. The second sentence puts weight on the faults (*fouten*), but they are willing to tolerate them, because of his merits.

1.3.4 What is aboutness?

In Information Retrieval (IR), the search for documents relevant to a given query is for the most part based on the notion of *aboutness*, which relies purely on the presence or absence of words in documents. Even though this approach is at first sight much less glamorous than semantic approaches, it is surprisingly effective. Semantic analysis of documents and queries is still experimental at best, but aboutness-based retrieval has been made highly practical by the introduction of word-based indexing techniques, the Vector Space Model and Language Modelling.

On the other hand, representing a document text as a bag of words destroys important information expressed in the phrases, the combinations of words used in the document. Intuitively it seems obvious that document representations capturing more of that information should lead to more accurate search. It should be made clear that when transducing a document into dependency triples, we are not striving for a representation of the meaning of a text, suitable for logical inference and reasoning, but rather for a representation of what is called in IR the aboutness of the text.

The notion of aboutness The notion of aboutness is highly central to Information Retrieval: the user of a retrieval system expects the system, in response to a query, to supply a list of documents which are about that query. Practical retrieval systems using words as terms are based on a surprisingly simpleminded notion of aboutness:

> If the word x occurs in the document then the document is *about* x.

This notion is then made less naive by introducing a measure for the similarity between the query and the document, and using other notions as term frequency and document frequency.

A model-theoretic basis for the notion of aboutness was described in [2]:

> An information carrier i will be said to be *about* information carrier j if the information borne by j holds in i.

The rather abstract sounding notion of "information carrier" can denote a single term, but also a composition of terms into a structured query or a document.

In other retrieval models (Boolean, vector space, logic-based or probabilistic) the notion of aboutness can be defined analogously [3].

The Aboutness Hypothesis According to Information Retrieval folklore, the best classification terms are the words from the open categories, in particular nouns and adjectives. The words from the closed categories are just stop words. In a classification experiment reported in [1], it was indeed found that nouns, verbs, and to a lesser degree adjectives and adverbs are the only words that contribute measurably to the aboutness of a text.

These observations lead us to the following Aboutness Hypothesis:

- of all the words in a text, the words from the open categories (nouns, verbs, adjectives, modifiers) carry most of the aboutness;
- of all possible dependency triples, the triples containing only words from the open categories will carry most of the aboutness;
- we expect a better classification result when using triples as terms in addition to words than when using words alone.

The elements of this hypothesis have been corroborated [10] in one of the hardest document classification tasks: the classification of patent documents. They use text categorisation as a indirect measurement of aboutness, because a well-defined measure lacks.

The aboutness of word categories Koster and Beney measured the contribution to the accuracy with one category of words, by classifying a given corpus using only words from that category as terms. The Parts-of-Speech, A, N, V and X stand for the open word categories: adjective, noun, verb and adverb.

	words	words + POS	A	N	V	X	A + N + V + X
aboutness	66.72	67.21	52.78	65.77	49.72	19.49	67.20

The results show that nouns have the highest aboutness, followed by adjectives and verbs.

The aboutness of relations The same has been done to determine the aboutness of relations. A relation consist of all dependency triples having the associated relator. For each relation they measured the contribution to the accuracy, and hence to aboutness.

	all triples	only ATTR	only OBJ	only SUBJ	only PREP	only of	only MOD
aboutness	66.15	67.26	58.52	56.65	58.98	56.22	41.56

The results show that the PREP relation turns out to be surprisingly important, next to the expected ATTR, OBJ and SUBJ relators.

2 The DUPIRA parser

The DUPIRA parser is a dependency parser and transducer for Dutch. The DUPIRA parser transduces factoids and noun phrases to dependency graphs following the aboutness-based model.

DUPIRA creates dependency graphs based on the aboutness-based dependency model. The traditional parse tree is a representation of the derivation process, a constructive proof that the sentence can be generated by the grammar. A dependency tree is a representation of the compositional structure of the sentence. Parse trees are full of irrelevant details, long, complex and highly dependent on the linguistic theory used; DUPIRA abstracts from these properties. It creates dependency trees that represent the main structure of a sentence in a compact way, while preserving the aboutness of the sentence.

In the aboutness-based dependency model, the main building blocks of a sentence are the phrases: the noun phrase, verb phrase, adjective phrase and adverb phrase, each being a generalisation by projection of a single word from an open category. DUPIRA represents the phrases with relations:

Phrase	Relations
noun phrase	SUBJ ATTR PREP PRED MOD DET QUANT
verb phrase	OBJ PREP MOD
adjective phrase	MOD
adverb phrase	MOD

All nodes of the dependency tree are marked with a content word (*non*-function word). All dependency triples have content words as head respectively modifier. A first estimate would be to the words from open categories. Additionally certain words from closed categories that are used like content words, such as personal pronouns (which act as nouns) and certain determiners (which act as adjectives).

2.1 Relations in the aboutness-based dependency model

In this subsection we will look at the relations per phrase, as they occur in the aboutness-based model.

2.1.1 Noun phrase

According to [14], a noun phrase is to be considered as a reference to, or description of a complicated concept. The noun phrase comprises a noun (the head) forming certain relations with its associated modifiers. It consists only of that noun in case there are no modifiers.

The noun phrase can contain the following relations:

- The SUBJ relation between a noun phrase occurring as a subject and the verb phrase of which it is the subject;
- The ATTR relation between the head noun and an adjective (or adjective phrase) specialising it;
- The PREP relation, parametrised with some preposition, between the head of the noun phrase and another noun phrase;

- The PRED relation between two nouns, expressing a claimed *is-a* relation between the NPs of which they are the heads;
- The MOD relation between the head and a (pre- or post-modifying) adverb;
- The DET relation between the head and some determiner;
- The QUANT relation between the head and a quantity.

2.1.2 Verb phrase

The term *verb phrase* is used for phrases that have a (main) verb as head, and a number of phrases (its *complements*) as modifiers. In Linguistics these are usually called *clauses* but the name verb phrase expresses more clearly a duality or complementarity between nouns and verbs.

Again according to Winograd, a verb phrase can be seen as the *description of a fact, event or process*. It describes something dynamic, in contrast to the noun phrase which describes something static. For the aboutness of the phrase, only the main verb is of importance, because the auxiliaries serve only to indicate time, modality or emotion. On the other hand, the relations between the main verb and its complements, including the subject, are essential for the aboutness of the phrase.

The verb phrase can contain the following relations:

- The OBJ relation between the head verb and its object (a noun phrase);
- The PREP relation, again parametrised with some preposition;
- The MOD relation between the verb and an adverb phrase.

The PRED relation is not attached to a verb in the case of copulae. Rather than saying

- [ship <SUBJ [be <PRED fast]]
  The ship is fast.

the aboutness-based model prefers

- [ship <PRED fast]
  The ship is fast.

because the copula belongs to a closed category, and the first expression leads to two triples [ship <SUBJ be] and [be <PRED fast] which carry by themselves little more aboutness than the individual words *ship* and *fast*.

2.1.3 Adjective phrase

The adjective phrase comprises an adjective with its associated adverbs (if any). The only relation is an adjective phrase is

- The MOD relation between the head adjective and an intensifier.

2.1.4 Adverb phrase

The adverb phrase consists of an adverb with possible intensifiers, which are indicated with MOD.

2.2 Syntactic and semantic analysis

In the previous subsection we saw which syntactic elements in the sentence are preserved in the transduction. In this subsection we will discuss the syntactic and semantic elements that get lost in the transformation from sentence to dependency graph according to the aboutness-based model. This section will not be an exhaustive list of aspects which are not preserved in the transformation, but it will indicate for the sorts of troubles which rise.

2.2.1 Elimination of words

The aboutness-based model prescribes we should only make dependency triples containing content words. This implies that all the non-content words are eliminated. These non-content words (often called function words) comprise articles, quantifiers, connectors &c. Removing these words amounts to applying a stop list, but removing words following the aboutness-based model is more rigorous; because also non-frequently occurring words that are non-content words get eliminated. Stop lists in general are about filtering meaningless words, stop lists in the context of the ABM, are about filtering words that do not add to the aboutness.

Examples of such non-content words are: *wie*, *wat*, *moeten* (if it functions as a auxiliary verb), *en* (the connector for two sentences), &c. Also the prepositions are removed, like *van* and *door*, because the prepositions themselves do not add to the aboutness, but they are stored as a parameter in the PREP relation.

2.2.2 Lemmatisation

Because of the normalised character of dependency triples, words are lemmatised. Both nouns and verbs are morphologically normalised. For nouns this means that all the plurals are converted to singulars, and for verbs this consists of removing their inflection and turning the verb into an infinitive.

In the process of changing a verb into its canonical form (the infinitive), grammatical categories can be affected, for example the categories: person, number, tense, aspect, voice and mood. For example: *Were the flowers planted by me?* After lemmatising, the result is: *I plant the flower.*

2.2.3 Embedded constructions

Although we only deal with factoids, embedded constructions like participle constructions and relative clauses might occur. From these embedded constructions new sentences are derived.

The man who wore a black velvet hat, was driving a car. The embedded construction (*who wore a black hat*), is connected to *man*, and can be used to create a new sentence: *The man wore a black velvet hat.* The original sentence can then be changed into: *The man was driving a car.*

In the original sentence it is obvious that it is about one man: a man that wore a black velvet hat. When separating the embedded construction from the main sentence, both sentences mention *the man*. But this need not imply it is about the same man. The determiner *the* does not necessarily refer to that specific man. An improvement would be: *The man was driving a car. He wore a black velvet hat.* But because this requires deep semantic parsing to resolve the anaphora, this is not supported by DUPIRA.

2.2.4 Adjective phrases

All kinds of specialised adjective phrases are not represented as modifiers to the head adjective. Examples of such adjective phrases are: *too good to be true*, *12 inch wide*, *as poor as a beggar*, &c. Such phrases serve to express only modalities which would require deep semantic analysis, and are therefore not exploited.

In the case of multiple adjectives specialising a noun, these adjectives are not considered nested, but they all separately specialise the noun: *a big red car* is both *a big car* and *a red car*.

2.2.5 Intensifiers

Adverb phrases consists of an adverb with possible intensifiers. The intensifiers that indicate emotion or modality are disposed, as they do not match to the definition of factoids. This means constructions, like *the most left car*, are ignored. Some other intensifiers are important to keep, negation being the most prominent example.

Compare: *He was a not very generous rival of Shelley* and *He was a very generous rival of Shelley*. We can remove *very*, because it indicates "merely" modality. The negation on the contrary, gives the noun *rival* a different property: generous versus not generous.

2.2.6 De-nouning of nominalisations

It is possible to transform nominalisations into corresponding verb phrases. It changes the phrase from focusing on objects or concepts into a phrase that is describing actions. Compare: *The improvement of the provision of ...* and *Improving the provision of ....*

3 Paraphrasis

In the previous section we discussed the transformation from sentence to dependency tree. In this section we will we look at the transformation the other way around: from dependency tree to sentence. We call this transformation paraphrasing a dependency tree.

3.1 How can a dependency tree be paraphrased?

The approach used in this thesis is to exploit the tree structure of the dependency graph. Using information about the parent of the node, its siblings and its ancestors, we have for each node some sort of context. For example, whether the node is embedded in a verb phrase, or is part of an adjective phrase. In our implementation we did not use information about the node's ancestors, but only from its direct parent. The dependency graphs are paraphrased in a recursive depth-first left-to-right way. The reason to use such a restricted context, is to see how well a simple algorithm would perform, and what limits would be reached. The results of having just a simple algorithm are discussed in the final section of this chapter.

Since the 80's, scientists have been researching word order. Dutch is a remarkable language [13] (especially in a European context), a lot of study has been done to find out about word order in Dutch. This is of great use, because languages have a tendency to be either left branching or right branching. According to [8], the position of the head, or the branching, is considered to be one of the main parameters of language variation.

In general the division between languages start with being either VO (verb-object) or OV. If a language is VO, it is likely that it is also a prepositional language (PN (preposition-noun)), and is NG (noun-genitive), NA (noun-adjective) and NRc (noun-relational clause).

Dependency graphs are not necessarily binary, a node can have an out-degree of zero, or more. In Dutch, words are often grouped according to a certain scheme: a combination of adverb - adjective - adverb is not possible if both adverbs are modifying the adjective. Nor is noun - numeral - noun. The possible variation for the first combination is: adverb - adverb - adjective; the second can not occur in Dutch. Knowing that there is a certain scheme to which the ordering holds on, we can try to generalise, it given a certain node-context.

If dependency trees were built with a binary branching method in mind, theories like head-driven phrase-structure grammars could be part of the solution. If there was a binary branching representation, we could paraphrase in a different manner. Thus when for each relation it is known whether it is left or right branching, it is simple to construct a natural looking sentence from the dependency tree. Alas, languages aren't always that consistent in their choices. Compare the three phrases below which express both a NG and two GN possibilities. NP denotes a noun phrase, GNP a genitive noun phrase, and G the genitive part of a GNP.

- NG: [$_{\text{GNP}}$ [$_{\text{NP}}$ Het huis] [$_{\text{G}}$ van mijn vader]]
- GN: [$_{\text{GNP}}$ [$_{\text{G}}$ Mijn vaders] [$_{\text{NP}}$ huis]]
- GN: [$_{\text{GNP}}$ [$_{\text{G}}$ Mijn vader zijn] [$_{\text{NP}}$ huis]]

In contrast to Spanish, which only has NG as possibility:

- NG: [$_{\text{GNP}}$ [$_{\text{NP}}$ La casa] [$_{\text{G}}$ de mi padre]]

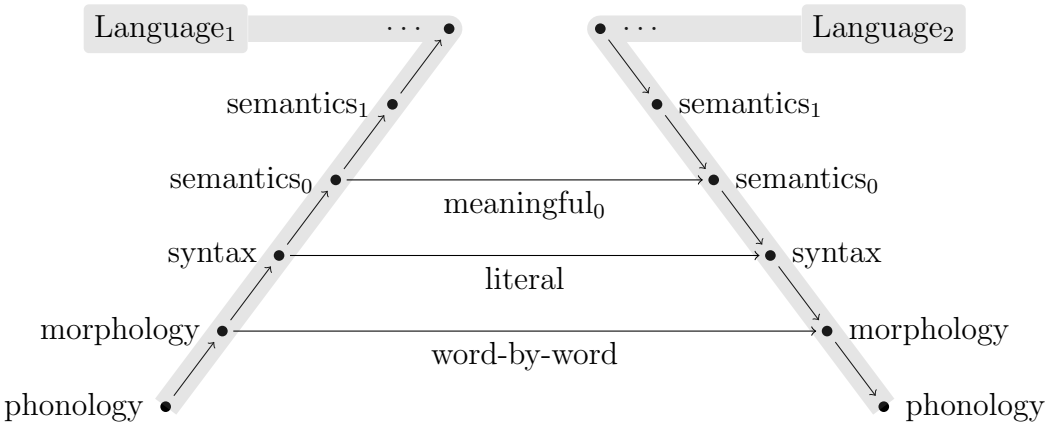
The DUPIRA parser translates all these genitive constructs with PREPvan, which means that when paraphrasing a dependency tree, you have to make choices about the word order, and the constructs you are going to use. For Spanish it is obvious in this case, for the Dutch sentence you can choose between the first or second option (the third is only possible in a spoken informal context). According to [7, 12], for Dutch, the most frequent word orders are:

Order of	Dominant order	DUPIRA
Subject, Object and Verb	No dominant order	SUBJ, OBJ
Subject and Verb	SV	SUBJ
Object and Verb	No dominant order	OBJ
Object, Oblique and Verb	No dominant order	MOD, PRED
Adposition and Noun Phrase	Prepositions	PREP
Genitive and Noun	Noun-Genitive	PREPvan
Adjective and Noun	Adjective-Noun	ATTR
Demonstrative and Noun	Demonstrative-Noun	DET
Numeral and Noun	Numeral-Noun	QUANT

Some of these classes very straightforward. For example, in Dutch the determiner is always left of the NP. As is the adjective in contemporary Dutch (viz. Middle Dutch: een ridder *koene*, and contemporary Dutch: een *koene* ridder).

We will only deal with the classes supported by the DUPIRA parser [9]: the noun phrase, verb phrase, adjective phrase and adverb phrase.

According to the model of Yngve and Bolliet [15], translations, and thus paraphrasis, can have several layers of depth.



Although the relations are all on a syntactic level, by using the concept of aboutness we try to do a *meaningful translation*. Meaningful in a sense that we preserve the aboutness. The source language, Language$_1$, is the language describing the original language (Dutch), and the destination language, Language$_2$, is the natural language we are translating to: Dutch. To translate from the source to destination language we use dependency graphs as transfer.

The following sections are about using meaningful$_0$, and using the information it provides to create sentences in Language$_2$ on a semantics$_0$ level. The result of this translation must be a factoid.

3.1.1 Noun phrase

Subject The subject relation is denoted with SUBJ. Because Dutch is a SVO language for main clauses, we can place the subject at the front. It doesn't matter if the subject is a singleton:

- [hij]<SUBJ [komen <MOD zeker <PREPaan [beurt <DET de]]
  Hij komt zeker aan de beurt.

or the subject is a more complex phrase, such as:

- [woorden <DET zijn]<SUBJ [vertederen <OBJ [ik]]
  Ik was vertederd door zijn woorden.

Adjective The ATTR relation between head noun and an adjective represents an adjective relation. In Dutch, adjectives always precede the corresponding noun:

- [hij]<SUBJ [streven <PREPnaar [sanering <ATTR voortgaande <DET een]]
  Hij streeft naar een voortgaande sanering.

- [kikvors <DET een]<SUBJ [zijn <PRED [kikker <ATTR groen <ATTR groot <DET een]]
  Een kikvors is een grote groene kikker.

Preposition The PREP relation is a relation parametrised with a preposition, between the head of the noun phrase and another noun phrase. Prepositions are, as is evident from the name, adpositions that are placed at the front of the other noun phrase that follows:

- MEN<SUBJ[tolereren<OBJ [fouten <DET zijn]<PREPvanwege [diensten<ATTR ver<DET zijn]]
  Men tolereert zijn fouten vanwege zijn verdiensten.

A prepositional relation is marked with the relator PREP with the type of preposition attached to the relator, i.e.: PREPaan, PREPvanwege, PREPdoor, &c.

Predicate The PRED relation, the predicate complement relation, between two nouns, expresses a *is-a* relation between the meanings of the NPs of which they are the heads. The predicate has a comparable function to the object, they only differ in their truth value [4]. The predicate should succeed the main verb, just as an object should, in a SVO language.

- [huis <DET mijn]<SUBJ [zijn <PRED [huis <DET jouw]]
  Mijn huis is jouw huis.

- [rover <ATTR dertiende<DET de]<SUBJ [heten <PRED [Ali Baba]]
  De dertiende rover heet Ali Baba.

Adverb The adverb is modelled as a modifier, denoted with MOD. The modifier relation holds between the head (i.e., another adverb, verb or adjective) and an adverb.

In Dutch two sub-categories of adverbs are discriminated. The first category contains the adjectives which are used as adverbial phrases. These adjectives cannot be inflected other than the comparative, superlative or diminutive. The second category contains the "real" adverbs. These are not to be inflected.

If an adverb is related to a verb, it succeeds the verb (cf. §3.1.2, as this can only occur in a verb phrase). If the adverb is used as an adverbial phrase (the first category), and if it is related to a adverb or adjective; it is put in front of that adverb or adjective (cf. §3.1.4).

An example of a adverb in a noun phrase:

- [[hond <MOD alleen] <SUBJ vragen <OBJ [zoiets]]
  Alleen honden vragen zoiets.

such adverbs are placed in front of the noun they modify, before the determiner and other modifiers.

Determiner The determiner relation is denoted with DET. The determiner always precedes the noun:

- [angst <DET de]<SUBJ[bekruipen <OBJ [me]]
  De angst bekruipt me.

All modifiers of the noun to which the determiner relates, are placed between the determiner and the noun:

- [uitzendbureau <ATTR kleine <DET het]<SUBJ [weten <MOD daar <PREPvan [niets]]
  Het kleine uitzendbureau weet daar niets van.

- [uitzendbureau <ATTR [kleine <MOD erg] <DET het]<SUBJ [weten <MOD daar <PREPvan [niets]]
  Het erg kleine uitzendbureau weet daar niets van.

Quantifier The quantifier acts like a determiner. The QUANT relation holds between the head and a quantity. The quantifier always succeeds the determiner if one is present, otherwise it will take the determiner's spot.

- [rovers <QUANT veertig]<SUBJ[zitten <PREProond [kampvuur <DET het]]
  Veertig rovers zitten rond het vuur.

3.1.2 Verb phrase

Object The object relation is denoted with OBJ. The relation is between the verb and a noun phrase, the object.

- [ik]<SUBJ[hebben <OBJ [[klap <PRED [gegeven]]<DET een] <PREPaan [hem]]
  Ik heb een klap gegeven aan hem.

Because Dutch is SVO, the object succeeds the verb.

Preposition Analogously to the prepositions in the noun phrase (§3.1.1).

Adverb If an adverb is related to a verb, it succeeds the verb:

- [ik]<SUBJ[houden <PREPvan [jou] <MOD daarom]
  Ik houd daarom van jou.

3.1.3 Adjective phrase

The adjective phrase comprises an adjective with its associated adverbs (if any).

Adverb The MOD relation between the head adjective and an intensifier. The intensifiers precede the adjective.

- [hij]<SUBJ[dragen <OBJ [jas <ATTR [groot<MOD erg]]<DET een]
  Hij draagt een erg grote jas.

3.1.4 Adverb phrase

The adverb phrase consists of an adverb with possible intensifiers. These intensifiers do not express nor emotions nor modality: the fact that the original sentence was a factoid rules this out.

- [feest <DET het]<SUBJ[beginnen <MOD daar]
  Het feest begon daar.

As adverbials can be treated as complements, they are, according to the definition of factoids, at the end of the sentence.

3.2 How can it be achieved that the paraphrase leads to a fluent and natural looking sentence?

By putting the words in the right order, one does not create a fluent and natural looking sentence. Within a sentence the words have mutual relations, which express themselves in agreements. These agreements are not always visible (i.e., pro-drop and PRO). If they are to be visible, we have to make them visible.

But sentences do not only feel natural when their syntactical elements are in place, and the agreements are coherent. The famous example from Chomsky proves this statement: "colourless green ideas sleep furiously," which is grammatical yet it feels ill formed because it has no meaning.

We have not focussed on meaning, but on aboutness. In the paraphrased sentence the aboutness is to be the same as in the original sentence. Therefore, when paraphrasing, we do not alter the aboutness, and thus we cannot prevent such "meaningless" sentences from being formed. On a syntactic level we are free to rearrange the words, which we did in the previous section. In this section we look at some techniques to give the sentence a more natural looking feeling.

3.2.1 Inflection

When a sentence lacks inflection, it does not look natural. In most cases people can understand the sentence, but since information is lacking, the reader's mind has to compensate for all the lack, resulting in both lower reading rates and the introduction of uncertainty.

In Dutch there are two sort of inflections: declension and conjugation. Some might argue conjugation takes place before declension, but it does not add extra value in this case, as we do not discuss this in great depth. So we stick to the order: declension before conjugation.

Declension The first step is to determine the number of the nouns. This information is not stored in the dependency graph, thus it should be retrieved from a context. The second step is to determine the role in the sentence. This depends on the verb, and the roles it can give away (i.e., subject, direct object, indirect object, &c.). In Dutch the nouns do not change according to their case, but the pronouns do.

Conjugation The conjugation of the verb(s) depends on the NPs that take the roles that the verb gave away. Besides the person and number, grammatical categories like time, aspect, voice and mood, depend on the context.

3.2.2 Word order

At first, paraphrasing looks like putting the words in the right order, and finish with adding missing information, like inflection. This fixed word order comes from the definition of factoids: subject – verb – object – complements (§1.3.2). Although word order is strict in many languages, it is by no means fixed in most of the languages. For example, in Dutch there is a rule which says that in a declarative sentence, the second constituent is a main verb: the verb-second rule. For declarative sentences Dutch is a SVO language. But this does not mean we cannot place another constituent in front: this is called inversion. As example we take another look at the sentence with the adverb phrase from §3.1.4.

- [feest <DET het]<SUBJ[beginnen <MOD daar]
  Daar begon het feest.
  Het feest begon daar.

begon is the main verb, and *daar* respectively *het feest* are the first constituents of the sentences. The second sentence is the default Dutch word order: SVO. The first sentence has been inversed. This inversion can be used to stress or topicalise the first constituent. In Dutch it is common to place adverbials that mention time or place, in front.

3.2.3 Co-reference chains

In both spoken and written text, people prefer to avoid literal repetition. This can be realised by choosing other words, another turn of phrase, anaphora, synonymy, hypernymy, &c. When a text is about *the black Siamese cat with a infected paw*, we do not repeat that complete phrase in every sentence, or even within the sentences. Repeated occurences can be: *the black Siamese cat*, *the cat with the infected paw*, *the cat*, *it*. All these references rely on the fact that the reader knows about the *the black Siamese cat with a infected paw* in a certain context.

3.2.4 Order of adjectives

In the Dutch language there are two rules for the order of adjectives [6]. These rules are not very strict and not using them does not lead to misconceptions. But they are part of the standard Dutch according to [6]. The first rule is the order of adjectives in a normal situation:

1. Adjectives which give a subjective judgement;
2. Adjectives which don't belong to the other classes;
3. Adjectives which mention the shape or colour;
4. Adjectives which mention the material.

The second rule is to put exceptions to the first rule:

- If an adjective is distinctive in a certain context, it is moved further to the left.

These rules imply that not only the meaning and the use of the adjectives have to be known, but also the context. This means an external source of information has to be used to determine the order.

In spoken text you can stress the distinctive adjective, and use the order like in a normal situation. But in written text, changing the order takes away ambiguity, which could otherwise rise. Compare:

- Een Rode Lori is een rode kleine papegaai;
- *Een Rode Lori is een kleine rode papegaai;
- Een Rode Lori is een rode kleine papegaai, een vasapapegaai is een grijze kleine papegaai.

4 Implementation

We know from the definition of dependency triples that a triple consists of a head and modifiers (if any). These modifiers can either precede or succeed the head. We use the word orders corresponding to relations as in the previous chapter. So for each node we divide its modifiers into those that are positioned in front, and those that are after the head. This is done recursively, such that for each head the modifier positions are sorted out.

The implementation is done in CDL3, a deterministic programming language based on affix grammars, intended for the development of compilers and other complicated pieces of software that can be implemented as syntax-directed transducers. The complete code listings are available in the appendices.

First, we check the syntax of the input sentence. This is done to prevent the program from outputting stange sentences because of invalid sentences. The next step was to parse the dependency tree. These two steps are implemented to be executed in one pass. If the dependency graph has a valid syntax, the parse tree of the dependency graph is built, and can be retrieved. If the dependency graph has an invalid syntax, the pass is interrupted to display an error to the user.

With the internal representation of the dependency graph, the transducer transduces the graph into a sentence with respect to word order: the canonical paraphrasis of factoids.

4.1 Results

The program has been tested on output from the DUPIRA parser. The implementation can successfully paraphrase all the dependency graphs used in the previous sections. The only thing that is missing are the features to make the paraphrase leads to a fluent and natural looking sentence, the features we discussed in §3.2. We present example output with all the relations from the aboutness-based model:

- MEN <SUBJ [verrichten <OBJ [groots<DET wat]<MOD daar]
  MEN verrichten daar wat groots
  Daar werd wat groots verricht.

- [je]<SUBJ [verpesten <OBJ [kansen <DET mijn] <PREPin [geval <DET ieder] <MOD daarmee]
  je verpesten daarmee mijn kansen in ieder geval
  Daarmee verpest je in ieder geval mijn kansen.

- [gedachten <ATTR somber] <SUBJ [overweldigen<OBJ [hij]<MOD daarna]
  somber gedachten overweldigen daarna hij
  Daarna werd hij door sombere gedachten overweldigd.

- [dat]<SUBJ [duren <OBJ [leven <ATTR heel<DET je] <MOD verder]
  dat duren verder je heel leven
  Dat duurt verder je hele leven.

- [show <DET die]<SUBJ [zijn <PRED ver <PREPvan [bed <DET mijn]]
  die show zijn ver mijn bed van
  Die show is ver van mijn bed.

- [wij]<SUBJ [geven <PREPaan [jou] <OBJ [erwtensoep] <PREPmet [worst] <PREPin [winter <DET de]]
  wij geven erwtensoep in de winter met worst aan jou
  In de winter geven wij jou erwtensoep met worst.

- [wij]<SUBJ [blijven <MOD daar <PRED [weken <QUANT twee]]
  wij blijven daar twee weken
  Wij bleven daar twee weken.

These results are very promising as the sentences feel very natural. This proves that with a simple algorithm decent results can be achieved.

From a random set 75 valid sentences, 69 sentences were paraphrased correctly. Among the 6 incorrectly paraphrased sentences are:

- [gedachten <ATTR somber] <SUBJ [overweldigen<OBJ [hij]<MOD daarna]
  somber gedachten overweldigen daarna hij
  Daarna werd hij door sombere gedachten overweldigd.

- [ik]<SUBJ [houden <PREPvan [jou] <MOD daarom]
  ik houden daarom jou van
  Daarom houd ik van jou.

- [ik]<SUBJ [meeleven <PREPmet [PSP <DET de] <MOD daarom]
  ik meeleven daarom met de PSP
  Daarom leef ik mee met de PSP.

Why these errors occur, is the subject of the next subsection.

4.2 Possible problems

Sometimes invalid paraphrases rise because of invalid input, more often because the information stored in a dependency graph does not provide all the details needed to paraphrase correctly. In our implementation we ruled out the first source of invalid paraphrases, which leaves us with the second source. While testing our implementation only one problem rose.

4.2.1 Separable verbs

In the last example of the previous subsection, the main verb is *meeleven*. As many verbs in Dutch, this is a separable verb. This raises two problems. The first problem is to recognise the verb as a separable verb. A *seperable verb* is a verb composed of a verb stem and a separable affix. Recognising a verb as a separable verb is not possible from just a dependency graph, and such information must therefore come from an external source. If *ge* is added to the infinitive when making the past participle (like *uitgelegd*) of a compound verb, it is always a separable verb. This way we can discriminate between *voorkomen* (to occur) and *voorkomen* (to prevent), respectively *voorgekomen* and *voorkomen*.

The second problem is to handle separable verbs. The verb stem of a separable verbs behaves just as a regular main verb, thus only the separable affix are the problem.

If we could decompose the verb, we could place the separable affix either at the end of the sentence or after the modifiers of the verb stem. Reconsider the example:

- [ik]<SUBJ [meeleven <PREPmet [PSP <DET de] <MOD daarom]
  Daarom leef ik mee met de PSP.
  Daarom leef ik met de PSP mee.
  Ik leef daarom met de PSP mee.
  Ik leef daarom mee met de PSP.

5 Discussion

In this thesis we showed how dependency trees can be paraphrased into factoids following the aboutness-based model. We started with four questions:

1. How can a factoid sentence be analysed into an aboutness-based dependency graph?
2. What syntactic and semantic aspects of the original sentence are lost in an aboutness-based dependency analysis?
3. How can the dependency tree deriving from a certain factoid be paraphrased into another factoid with the same aboutness?
4. How can it be achieved that this paraphrase leads to a fluent and natural looking sentence?

5.1 Summary

Each of the questions was another step towards a paraphrasis of a dependency graph. We will now conclude each question and we look for the possibilities for further research.

5.1.1 How can a factoid sentence be analysed into an aboutness-based dependency graph?

In section 1 we defined the aboutness-based model and the terms factoid and dependency graph. In section 2 we presented the DUPIRA parser which analyses factoid into aboutness-based dependency graphs according to the aboutness-based model.

5.1.2 What syntactic and semantic aspects of the original sentence are lost in an aboutness-based dependency analysis?

In section 2 we did not only look at the aspects that are preserved in the analysis, but we also looked at the syntactic and semantic aspects that are lost in the analysis. Not every aspect looks as noticeable as the others. The aspect that, when removed, is most noticeable is the lemmatisation: the removal of declension and conjugation. This means that the original sentence can not be recovered by just the information stored in the dependency graph.

When comparing the words in the dependency graph with the words in the original factoid, there is often a discrepancy. This semantic loss is due to the application of stop lists, the removal of adjective phrases and intensifiers that indicate emotion or modality, and transforming embedded constructions into new factoids. Removing words also influences the syntactic structures in the dependency graph compared to the syntactic relations in the original factoid.

5.1.3 How can the dependency tree deriving from a certain factoid be paraphrased into another factoid with the same aboutness?

In the section Paraphrasis we introduced the concept of word order. For almost any language the dominant word orders are known for several syntactic constructs, basic word order being one of the most famous. This information about dominant word order can be

used in the process of transducing a dependency tree into a factoid. The strategy is to determine for each relation for each phrase (noun phrase, verb phrase, adjective phrase, adverb phrase) the dominant order. The modifiers are then divided in two classes: the words that precede the head of the phrase, and the words that succeed the head of the phrase. While dividing the modifiers in one of the two classes, we keep in mind the preferred order from the words within a class. For example, [7] says that in Dutch both determiners and adjectives precede the noun. It does not say whether the determiner should precede or succeed the adjectives, when both a determiner and a adjective are modifying the head. In the sections 3.1.1 – 3.1.4 we showed the order of the words for each relation in the aboutness-based model, together with their ordering when there are multiple modifiers to modify the head.

The transduction of a factoid into a dependency tree is a deterministic process, and multiple sentences can map onto one dependency tree. The syntactic relations of the dependency tree are then used to paraphrase. Because the paraphrasis is solely based on these syntactic relations, the transduction of the paraphrase yields the same dependency tree as the transduction of the original factoid. And because the dependency tree is the same, the aboutness is the same.

5.1.4 How can it be achieved that this paraphrase leads to a fluent and natural looking sentence?

The first step to create a sentence that is fluent and looks natural, is to create a grammatical sentence: the words are in the right order, and the words are inflected. A grammatical sentence can still look unnatural, for example: if the multiple adjectives that modify a head are not in the right order. It is syntactic correct to place the adjectives in front on the head, but the human reader prefers, or rather, expects, the adjectives to be in a specific order.

Although the term word order sounds like the order is fixed, natural looking sentences have a variation in the word order. This can be done to avoid repetition, but can also be used to stress certain constituents within a sentence or discourse.

5.2 More advanced paraphrasing

The DUPIRA parser transduces factoids to dependency graphs. This transduction is a deterministic process based on the aboutness of the factoid, and can be considered a many-to-one mapping. The paraphrasis described in this thesis is also deterministic: the same dependency graph always yields the same result.

Just as word order within sentences makes that the text feels more natural, so is word variation on a lexical level. A way to achieve variation is to create a non-deterministic paraphraser, with a one-to-many mapping. In an ideal situation the statistics of all the possible input factoids are known, such as word frequency, which synonyms are interchangeable in a certain context, &c. For word order this can for example be determined by looking at the chance that a specific dependency triple occurs. The non-deterministic paraphraser should then yield factoids that have the same statistical distribution as the input factoids.

One way to make sure the inflection, word order and other aspects that give the sentence a natural look, is to incorporate these aspects in the grammar. This grammar is an inverted grammar, used to analyse a dependency graph into factoids. This is in

contrast to the non-inverted grammar, which is used to analyse factoids into dependency graphs. The grammar can de-lemmatise the words with respect to predicates, such that words that are not case-invariant (such as pronouns) and verbs that are not affix-invariant, are inflected with respect to the other relations in the sentence.

5.3 Further research

This thesis only investigates a canonical paraphrasis of factoids. Further research is needed to create more natural looking and fluent sentences with a non-deterministic paraphraser with the use of an inverted grammar.

6 Conclusion

In this thesis we have investigated the paraphrasing of factoid dependency trees into fluent sentences in a natural language. If we limit ourselves to the relations in the dependency trees described in the aboutness-based model, we can achieve promising results. We find that the words that are eliminated in the analysis into a dependency graph are not a great loss, because the overall aboutness of the factoid is preserved. The removal of inflection, on the contrary, removes so much information that aspects such as plurality and tense are lost. This cannot be retrieved without a context. The simple paraphrasis as described in this thesis cannot create fluent and natural looking sentences, because too much information gets lost. However, the results from this thesis can be used as a benchmark for future systems.

References

[1] Avi Arampatzis, Th. P. van der Weide, C. H. A. Koster, and P. van Bommel. An Evaluation of Linguistically-motivated Indexing Schemes. In *In Proceedings of the 22nd BCS-IRSG Colloquium on IR Research*, 2000.

[2] P. D. Bruza and T. W. C. Huibers. Investigating Aboutness Axioms using Information Fields. In *In Proceedings of ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 112–121. Springer-Verlag, 1994.

[3] P. D. Bruza and T. W. C. Huibers. A Study of Aboutness in Information Retrieval. *Artificial Intelligence Review*, 10:1–27, 1996.

[4] Robyn Carston. The Semantics/pragmatics Distinction: A View From Relevance Theory. In *UCL Working Papers in Linguistics 7. 1-26.*, pages 1–30. Elsevier Science, 1998.

[5] Mark Dras. *Tree Adjoining Grammar and the reluctant Paraphrasing of Text*. PhD thesis, Macquarie University, NSW 2109 Australia, February 1999.

[6] K. Romijn G. Geerts J. de Rooij Haeseryn, W. and M.C. van den Toorn. *Algemene Nederlandse Spraakkunst*. Martinus Nijhoff uitgevers/Wolters Plantyn, Groningen/Deurne, Tweede, geheel herziene druk edition, 1997.

[7] Dryer Matthew S. Gil David Haspelmath, Martin. *The World Atlas of Language Structures Online*, volume chapter 39 of *Max Planck Digital Library*. Munich, 2008.

[8] Jacob Hoeksema. Head-Types in Morpho-syntax. *Yearbook of Morphology*, 1:123–137, 1988.

[9] C. H. A. Koster. Aboutness-based Dependency Grammars for IR applications. Draft, October 2009.

[10] Cornelis H. A. Koster and Jean G. Beney. Phrase-based document categorization revisited. In *PaIR 09: Proceeding of the 2nd international workshop on Patent information retrieval*, pages 49–56, New York, NY, USA, 2009. ACM.

[11] Jens Nilsson, Joakim Nivre, and Johan Hall. Graph transformations in data-driven dependency parsing. In *ACL-44: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 257–264, Morristown, NJ, USA, 2006. Association for Computational Linguistics.

[12] William Z. Shetter. *Introduction to Dutch*. Martinus Nijhoff, The Hague, 1958.

[13] Gunther De Vogelaer. *Extending Hawkins comparative typology: Case, word order, and verb agreement in the Germanic languages*. PhD thesis, Ghent University, 2007.

[14] Terry Winograd. *Language as a cognitive process*. Addison Wesley Pub. Co., Reading, Mass., 1983 .

[15] Victor H. Yngve. Sentence-for-sentence translation. In *Mechanical Translation*, pages 29–37. Butterworths Scientific Publications, November 1955.

A Code

A.1 paraphraser.cdl3

```
ROOT paraphraser.
USES dg, transducer.

ACTION paraphraser:
  read line(TEXT),
    is valid syntax(TEXT, DG, INT),
    write result(INT),
    transduce dependency graph (DG, TEXT1),
    write ("\n");
  write("no word\n").

ACTION write result(>INT):
  [INT->1], write("DG is valid\n");
  write ("DG is invalid\n").
```

A.2 dg.cdl3

```
MODULE dg = STDERR DEPGR STATE.

DEFINES DG, HEAD, TERMS, TERM, MODS, MOD, REL,
        ACTION is valid syntax(>TEXT, DG>, RESULT>),
        ACTION error (>TEXT).

STDERR = FILE.
RESULT, STATE = INT.
C = TEXT.

DEPGR,DG= HEAD MODS.
HEAD = TERMS.
TERMS :: TERM and TERMS; empty.
TERM :: TEXT; embedded DG.
MODS :: MOD and MODS; empty.
MOD = REL TERMS.
REL = TEXT TEXT.

PRELUDE init dg:
  [empty empty -> DEPGR],
    [1 -> STATE] ,
    ( open standard error (STDERR);
        ?
    ).

ACTION is valid syntax(>TEXT>, DG>, RESULT1>):
  is dg(TEXT, DG, RESULT), [STATE -> RESULT1];
```

```
  [0 -> RESULT1], [empty empty -> DG].

PRED is dg (>TEXT>, DG>, RESULT>):
  is square open symbol(TEXT),
    should be head(TEXT, HEAD),
    may be modifiers(TEXT, MODS),
    should be square close token(TEXT),
    [HEAD MODS -> DG],
    [1->RESULT];
  is curly  open symbol(TEXT),
    should be head(TEXT, HEAD),
    may be modifiers(TEXT, MODS),
    should be curly  close token(TEXT),
    [HEAD MODS -> DG],
    [1->RESULT].

ACTION should be head(>TEXT>, HEAD>):
  is terms(TEXT, TERMS), [TERMS -> HEAD];
  error("ERR head"), [empty -> HEAD].

PRED is terms(>TEXT>, TERMS>):
  is term(TEXT, TERM),
    ( is bar symbol(TEXT),
      should be terms(TEXT, TERMS),
      [TERM and TERMS -> TERMS];
      [TERM and empty -> TERMS]
    ).

PRED is term(>TEXT>, TERM>):
  is quote open symbol(TEXT),
    is word (TEXT, WORD),
      should be quote close token(TEXT),
      [WORD -> TERM],
      skip blanks(TEXT);
  is dg(TEXT, DG, RESULT),
    [embedded DG -> TERM],
    skip blanks(TEXT);
  is word(TEXT, WORD),
    [WORD-> TERM],
    skip blanks(TEXT).

ACTION may be modifiers(>TEXT>, MODS>):
  is modifier(TEXT, MOD),
    may be modifiers(TEXT, MODS),
    [MOD and MODS -> MODS];
  [empty -> MODS].

PRED is modifier(>TEXT>, MOD>):
```

```
  is relator(TEXT, REL),
    should be terms(TEXT, TERMS),
    [REL TERMS -> MOD].

PRED is relator(>TEXT>, REL>):
  is direction left symbol(TEXT),
    ( is word(TEXT, TEXT1),
      ["<" TEXT1 -> REL],
      should be white space(TEXT);
      error ("Missing relator\n"),
        [" " " " -> REL]
    );
  is direction right symbol(TEXT),
    ( is word(TEXT, TEXT1),
      [">" TEXT1 -> REL],
      should be white space(TEXT);
      error ("Missing relator\n"),
        [" " " " -> REL]
    ).

ACTION should be terms(>TEXT>, TERMS>):
  is terms(TEXT, TERMS);
  [empty -> TERMS].

TEST is direction left symbol(>TEXT>):
  is prefix("<", TEXT, TEXT).

TEST is direction right symbol(>TEXT>):
  is prefix(">", TEXT, TEXT).

PRED is square open symbol(>TEXT>):
  is prefix("[", TEXT, TEXT), skip blanks (TEXT).

PRED is square close symbol(>TEXT>):
  is prefix("]", TEXT, TEXT), skip blanks (TEXT).

ACTION should be square close token(>TEXT>):
  is square close symbol(TEXT);
  is white space symbol(TEXT),
    skip blanks(TEXT),
    should be square close token(TEXT);
  error ("Close symbol ] expected\n"),
    ["ERROR" -> TEXT].

PRED is bar symbol(>TEXT>):
  is prefix("|", TEXT, TEXT), skip blanks (TEXT).

PRED is curly open symbol(>TEXT>):
```

```
  is prefix("{", TEXT, TEXT), skip blanks (TEXT).

PRED is curly close symbol(>TEXT>):
  is prefix("}", TEXT, TEXT), skip blanks (TEXT).

ACTION should be curly close token(>TEXT>):
  is curly close symbol(TEXT);
  error ("Close symbol } expected\n"), ["ERROR" -> TEXT].

TEST is quote open symbol(>TEXT>):
  is prefix("\"", TEXT, TEXT).

TEST is quote close symbol(>TEXT>):
  is prefix("\"", TEXT, TEXT).

ACTION should be quote close token(>TEXT>):
  is quote close symbol(TEXT);
  error ("Close symbol \"\n"), ["ERROR" -> TEXT].

TEST is white space symbol(>TEXT>):
  is prefix(" ", TEXT, TEXT).

ACTION should be white space(>TEXT>):
  is white space symbol(TEXT);
  error ("Some white space is missing after a relator\n"),
    ["ERROR" -> TEXT].

ACTION error (>TEXT):
   write (STDERR, TEXT + "\n"), [0 -> STATE].

WORD = TEXT.
PRED is word(>TEXT>, WORD2>):
  is letter(TEXT, WORD),
    word tail(TEXT, WORD1),
    [WORD+WORD1 -> WORD2].

ACTION word tail(>TEXT>, WORD3>):
  is letter(TEXT, WORD1),
    word tail(TEXT, WORD2),
    [WORD1+WORD2 -> WORD3];
  ["" -> WORD3].

TEST is letter(>TEXT>, WORD>):
  prefix(TEXT, 1, WORD, TEXT1),
    between(WORD, 0, "a", "z"),
    [TEXT1 -> TEXT];
  prefix(TEXT, 1, WORD, TEXT1),
    between(WORD, 0, "A", "Z"),
```

```
    [TEXT1 -> TEXT].

ACTION skip blanks(>TEXT>):
  is prefix(" ", TEXT, TEXT), skip blanks(TEXT);
  +.
```

A.3 transducer.cdl3

```
MODULE transducer = CURR_PHRASE.

DEFINES STRING,
        ACTION transduce dependency graph (>DG, STRING>).

USES dg.

STRING = TEXT.
IMODS = MODS.
OMODS = MODS.
RMODS = MODS.
IREL = REL.
PHRASE :: np; vp; nothing.
CURR_PHRASE = PHRASE.

PRELUDE fill blanks:
  [ np -> CURR_PHRASE ].

ACTION transduce dependency graph (>DG, STRING>):
  print dg (DG), ["" -> STRING].

ACTION print dg (>DG):
  [DG -> HEAD MODS], [HEAD -> TERMS],
    [empty -> MODS1], [empty -> MODS2],
    sort arguments(MODS, MODS1, MODS2),
    reverse(MODS1, RMODS1), print mods(RMODS1),
    print head(TERMS),
    reverse(MODS2, RMODS2), print mods(RMODS2);
  [DG -> HEAD MODS],
    [HEAD -> TERMS],
    print head (TERMS),
    print mods (MODS).

PRED sort arguments (>IMODS, >OMODS>, >OMODS1>):
  [CURR_PHRASE -> vp],
    should take modifiers (IMODS, "<" "MOD", OMODS1),
    should take modifiers (IMODS, "<" "DET", OMODS),
    should take modifiers (IMODS, "<" "QUANT", OMODS),
    should take modifiers (IMODS, "<" "ATTR", OMODS),
    should take modifiers (IMODS, "<" "SUBJ", OMODS1),
```

```
    should take modifiers (IMODS, "<" "OBJ", OMODS1),
    should take modifiers (IMODS, "<" "PRED", OMODS1),
    should take modifiers (IMODS, "<" "PREPbij", OMODS1),
    should take modifiers (IMODS, "<" "PREPals", OMODS1),
    should take modifiers (IMODS, "<" "PREPop", OMODS1),
    should take modifiers (IMODS, "<" "PREPin", OMODS1),
    should take modifiers (IMODS, "<" "PREPdoor", OMODS1),
    should take modifiers (IMODS, "<" "PREPmet", OMODS1),
    should take modifiers (IMODS, "<" "PREPnaar", OMODS1),
    should take modifiers (IMODS, "<" "PREPvanwege", OMODS1),
    should take modifiers (IMODS, "<" "PREPaan", OMODS1),
    should take modifiers (IMODS, "<" "PREProond", OMODS1),
    should take modifiers (IMODS, "<" "PREPvan", OMODS1);
  take modifiers (IMODS, "<" "DET", OMODS),
    should take modifiers (IMODS, "<" "QUANT", OMODS),
    should take modifiers (IMODS, "<" "ATTR", OMODS),
    should take modifiers (IMODS, "<" "SUBJ", OMODS1),
    should take modifiers (IMODS, "<" "OBJ", OMODS1),
    should take modifiers (IMODS, "<" "PRED", OMODS1),
    should take modifiers (IMODS, "<" "PREPbij", OMODS1),
    should take modifiers (IMODS, "<" "PREPals", OMODS1),
    should take modifiers (IMODS, "<" "PREPals", OMODS1),
    should take modifiers (IMODS, "<" "PREPop", OMODS1),
    should take modifiers (IMODS, "<" "PREPin", OMODS1),
    should take modifiers (IMODS, "<" "PREPdoor", OMODS1),
    should take modifiers (IMODS, "<" "PREPmet", OMODS1),
    should take modifiers (IMODS, "<" "PREPnaar", OMODS1),
    should take modifiers (IMODS, "<" "PREPvanwege", OMODS1),
    should take modifiers (IMODS, "<" "PREPaan", OMODS1),
    should take modifiers (IMODS, "<" "PREProond", OMODS1),
    should take modifiers (IMODS, "<" "PREPvan", OMODS1),
    should take modifiers (IMODS, "<" "MOD", OMODS).

ACTION should take modifiers (>IMODS>, >IREL, >OMODS>):
  take modifiers (IMODS, IREL, OMODS);
  +.

PRED take modifiers (>IMODS>, >IREL, >OMODS>):
  [IMODS -> MOD and MODS], [MOD -> REL TERMS],
    ([REL = IREL],
      [MODS -> IMODS],
      [MOD and OMODS -> OMODS],
      take modifiers (IMODS, IREL, OMODS);
     take modifiers (MODS, IREL, OMODS)
    );
  [IMODS -> empty], +.

ACTION print head (>TERMS):
```

```
  [TERMS -> TERM and TERMS1],
    print term(TERM),
    print head (TERMS1);
  [TERMS -> empty],
    write ("");
  error ("invalid TERMS").

ACTION print mods (>MODS):
  [MODS -> MOD and MODS1],
    print mod (MOD),
    print mods (MODS1);
  [MODS -> empty],
    write ("");
  error ("invalid MODS").

ACTION print rel (>REL):
  split prep (REL, TEXT1, TEXT2), write (TEXT2 + " ");
  [REL -> TEXT TEXT1], write("").

FUNCTION reverse (>IMODS, OMODS>):
   reverse (IMODS, empty, OMODS).

FUNCTION reverse (>IMODS, >MODS, OMODS>):
   [ IMODS -> MOD and MODS2 ], [ MOD and MODS -> MODS3 ],
      reverse (MODS2, MODS3, OMODS);
   [ IMODS -> empty ], [ MODS -> OMODS ];
   ?.

TEST split prep (>REL, TEXT1>, TEXT2>):
  [REL -> TEXT TEXT1], is prefix ("PREP", TEXT1, TEXT2).

ACTION print mod (>MOD):
  [MOD -> REL TERMS], change phrase (REL),
    ( is postposition(REL),
      print head (TERMS),
      print rel (REL)
      print rel (REL), print head (TERMS)
    ).

ACTION change phrase (>REL):
  [ REL -> "<" "SUBJ" ], [vp -> CURR_PHRASE ];
  [np -> CURR_PHRASE].

TEST is postposition(>REL):
  split prep (REL, TEXT1, TEXT2), [TEXT2 -> "van"].

ACTION print (>MODS):
  [MODS -> MOD and MODS1],
```

```
    print mod (MOD),
    print mods (MODS1);
  [MODS -> empty], write ("");
  error ("invalid MODS").

ACTION print term (>TERM):
  [TERM -> embedded DG], print dg (DG);
  [TERM -> TEXT], write (TEXT), write (" ");
  error ("invalid TERM").
```