PROS

AUTOMATIC PROSODIC SENTENCE ANALYSIS, ACCENTUATION AND PHRASING FOR DUTCH TEXT-TO-SPEECH CONVERSION

Hugo Quené and René Kager

FINAL REPORT

januari 1990

SPIN/ASSP Report nr. 17

Research Institute for Language and Speech
Rijksuniversiteit Utrecht
Trans 10, NL-3512 JK Utrecht
the Netherlands

CONTENTS

VOORWOORD

Gedurende het hier beschreven onderzoek, alsmede bij de verslaglegging daarvan, hebben we van velen inspiratie, hulp en adviezen ontvangen. De volgende personen willen we daarvoor bedanken:

- Vincent van Heuven, Marcel van den Broecke, Bert Schouten en Sieb Nooteboom, die als begeleiders bij dit onderzoek betrokken waren;
- Renée van Bezooijen, voor de door haar uitgevoerde perceptieve evaluatie;
- Philip Bloemendal, voor het inspreken van het spraak-materiaal;
- Sidonne Bos, Eldrid Bringmann, Yvonne van Holsteijn, Mariken ter Keurs, Jeroen Reizevoort en Ton Veenhof, die ons als student-assistenten veel werk uit handen hebben genomen;
- de (huidige en voormalige) leden van de themagroepen 'Taalkundige Analyse' en 'Prosodie', voor de vele discussies en raadgevingen tijdens de voortgang van het onderzoek;
- Henny Bekker (Facultaire Automatiseringsdienst), die de noodzakelijke computer-faciliteiten verzorgde;
- Ad Aerts (MediaSystemen, Haarlem), voor de grote bestanden kranteberichten die hij ons regelmatig leverde;

Utrecht, januari 1990

1. INTRODUCTION

Adequate prosodic cues make a sentence easier to perceive and to comprehend (e.g. Collier and 't Hart 1975; Wingfield 1975; Nooteboom, Brokx and De Rooij 1978; Cutler 1982; Nooteboom 1985; Cutler and Clifton 1984; Nooteboom and Kruyt 1987). Using prosodic cues, the listener can extract the linguistic structure of the message. *Pauses* between (linguistically) coherent word groups, for example, help the listener in three ways (Scharpff and Van Heuven 1988). Firstly, word segmentation is facilitated, because pauses coincide with word boundaries; the positive marking of these word boundaries in the speech stream reduces the ambiguity with regard to word segmentation. Secondly, the continuous speech signal is divided into coherent word groups or "chunks" of acoustic-phonetic information, which increases the intelligibility of the speech signal. Thirdly, the listener is provided with some extra processing time between these coherent word groups. Likewise, *accentuation* guides a listener's attention to the words which are considered important by the speaker, and which are acoustically most reliable.

These perceptual functions of sentence prosody become even more important when the speech signal is less redundant, providing fewer segmental cues to the intended speech sounds, words, and meaning. Obviously, artificial speech (such as the output of a text-to-speech conversion system) lacks the normal degree of acoustic-phonetic redundancy, because it is not known which (and how) phonetic details should be implemented. Consequently, correct prosodic cues may significantly improve the perception and comprehension of synthetic speech. Text-to-speech systems should aim at producing a natural sentence prosody, in order to compensate for their reduced speech quality.

In the production of natural speech, several prosodic phenomena depend on the intended *phrasing* of a speech utterance: a linguistic boundary may be marked by means of an appropriate F_0 movement (Collier and 't Hart 1975; Cooper and Sorensen 1977), a silent interval (Goldman-Eisler 1972), lengthening of the preceding speech sounds (Klatt 1975, 1976), blocking of coarticulation and sandhi (Cooper and Paccia-Cooper 1980), etc. These phenomena can be seen as *phonetic* correlates of the *abstract* notion "phrase boundary". Together, they indicate the intended division of the speech utterance into phrases.

Similarly, several prosodic phenomena depend on the *accent* of a word, i.e., its relative prominence. A speaker can convey this word prominence by means of appropriate F_0 movements (Collier and 't Hart 1975), in combination with higher intensity (Lehiste 1970), longer duration (Klatt 1975, 1976), and less vowel reduction (of the stressed syllable within the accented word; Koopmans-Van Beinum 1980). Again, these phenomena can be seen as phonetic correlates of the more abstract notion "accent" (Nooteboom and Kruyt 1987).

Thus, various phonetic aspects of sentence prosody depend (to a great extent) on the abstract linguistic notions *phrasing* and *accent*. From these latter two, a text-to-speech system can (in theory) derive many suprasegmental phenomena in the output speech: segmental durations, location and duration of silent intervals, F_0 movements (e.g. Van Wijk and Kempen 1985), vowel reduction, intensity pattern, sandhi and coarticulation, etc. Consequently, a high-quality text-to-speech system should attempt to establish both abstract prosodic phenomena, and to 'interpret' these into correct suprasegmental phenomena.

In natural speech, the produced (abstract) phrasing and accentuation are assumed to be related to the linguistic structure of an utterance. According to nonlinear phonological theory, sentence prosody does not depend directly on the syntactic surface structure, but rather on the related *prosodic sentence structure* (Nespor and Vogel 1982, 1986; Gee & Grosjean 1983; Selkirk 1984, 1986). In addition, focus structure and thematic structure[1] also affect the abstract prosody of a sentence (Gussenhoven 1984; Baart 1987).

Extending this line of though, we assume that accentuation and phrasing can both be derived from the prosodic sentence structure, provided that focus information and thematic structure are taken into account. To illustrate matters, our view of the various levels of sentence prosody is represented in Figure 1.

Figure 1: Three levels of sentence prosody.

linguistic:	prosodic structure, focus structure, thematic structure
abstract prosody:	phrasing and accentuation
phonetic prosody:	segmental durations, F_0 movements, pausing, intensity, vowel reduction, coarticulation, sandhi, etc.

The algorithm PROS, which is described in this report, aims at establishing the 'abstract prosody' for text-to-speech conversion. Phrasing and accentuation are derived from the prosodic sentence structure, in combination with focus information and thematic sentence structure. Firstly, a hybrid prosodic sentence structure is established (hence *PSS*). Although the resulting structure comes close to the prosodic sentence structure proposed by Nespor and Vogel (1982, 1986; explained below in more detail), thematic relations are also taken into account: separate thematic constituents (viz. Predicates, Arguments, Modifiers) usually correspond to separate prosodic domains. Subsequently, accentuation and phrasing are derived by means of this sentence structure (as well as by means of additional information) and inserted as abstract prosodic markers into the text sentence. The algorithm is intended as one of the many components in a Dutch text-to-speech system (Te Lindert, Doedens, and Van Leeuwen 1989). Other components convert the output (viz. 'abstract prosody') to adequate phonetic prosody: on the basis of the PROS output, 'silence' segments are inserted in the phoneme string, and the F_0 contour is calculated ('t Hart and Collier 1975; Van Wijk and Kempen 1985).

[1] The *thematic* sentence structure specifies which semantic constituents function as Predicate, Argument and Modifier [=Condition] (Gussenhoven 1984).

2. PROSODIC STRUCTURE: THEORY

From many languages, it is known that sandhi rules (i.e., rules of phonological adjustment between words) have their own specific domains of application. These domains are not necessarily isomorphous to syntactic constituents. Among others, Nespor and Vogel (1982, 1986) and Selkirk (1984, 1986) have made proposals as to the mapping between syntactic constituents and *prosodic domains*. Two prosodic domains attested in this sense are the phonological phrase (Phi) and the intonational phrase (Int). These domains are part of a hierarchical tree structure, viz. the prosodic sentence structure (PSS).

In the definition of these domains, the distinction between content words and function words plays a crucial role. *Content words* (CWs) carry the main semantic load of a sentence ('lexical words': Nouns, Verbs, Adjectiva and Adverbs). This class is extendable: new words are almost always CWs [e.g., English **aspirin** (N) or Dutch **jofel** (A)]. *Function words* (FWs) express the relations between the content words ('non-lexical words': Prepositions, Conjunctions, Complementizers, Copula, etc.). FWs have hardly any independent meaning; the class of FWs is fixed.

In languages such as English and Dutch, the *Phi* domain (or phonological phrase) is built around a lexical head, i.e., a CW which is the head of a syntactic constituent. It includes left-hand specifiers of the lexical head (either CWs or FWs), as well as all left-hand FWs. The (final) lexical head of each Phi is the *prosodic head*; this word plays an important role in accentuation (see section 3.3. below).

The next higher prosodic constituent is the *Int* domain (or intonational phrase). This constituent is constructed by grouping adjacent Phi domains. Hence, a whole Phi domain is always contained within a single Int domain. In addition, however, important syntactic breaks are also respected. In general, each syntactic constituent which is attached to any S-node (in the syntactic surface structure) establishes a separate Int domain. Consequently, [1] displaced syntactic constituents, [2] (most) subordinate clauses, and [3] parentheticals, are all separate Int domains.

the following example, the Phi and Int domains are illustrated in a flat represer ɔn (where '##' indicates an Int-boundary, and '#' a Phi-boundary). These exar :learly demonstrate that prosodic domains do not necessarily correspond to s constituents.

) ## Kasyapa's great war elephant # turned aside
## to avoid # a patch # of marshy ground ##

## de computer # spreekt # tot de bemanning
## op de betweterige # en begrijpende toon
## die we kennen # uit de zachte sector ##

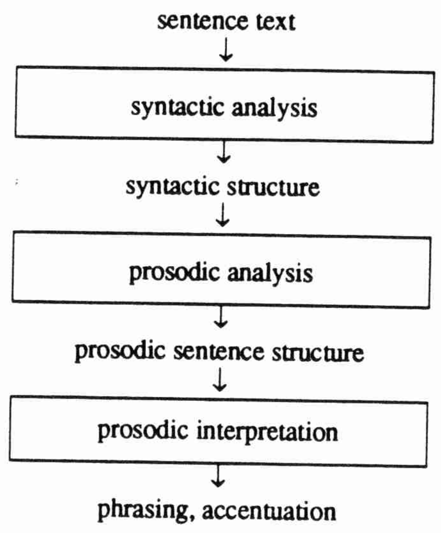
domains tend to be of equal length, and their length increases in faster :count for these effects, separate rules restructure the prosodic domains. ule joins a Phi consisting of only the lexical head to the Phi to its left, yntactic conditions. Very short Int's can be eliminated by merging them Int's, and very long Int's are broken down into shorter ones.

3. AUTOMATIC PROSODIC ANALYSIS

3.1 introduction

According to the linguistic theory described above, the prosodic structure is derived from the syntactic (surface) sentence structure. If a text-to-speech system needs to perform a prosodic analysis, then a syntactic analysis is also necessary (see Figure 2).

Figure 2: Linguistic method for deriving the prosodic sentence structure (PSS) from an intermediate syntactic surface structure.



However, this linguistically motivated method cannot be applied to automatic prosodic sentence analysis. Firstly, there is no algorithm for syntactic analysis (parser) available which performs satisfactorily for our purposes. Such a parser must be able to analyse any text, at a speed which exceeds the average speaking rate. This task requires a large set of syntactic rules, as well a large lexicon. At this moment, such a system is not (yet) available for Dutch.

Secondly, if such a parser did exist, it would run into great difficulties when analysing syntactically ambiguous sentences like the following:

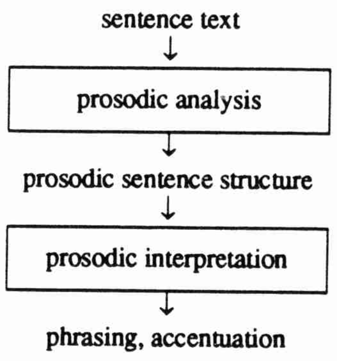
(2a) I (have mown) (the lawn with the flowers)
(2b) * I (have mown) (the lawn) (with the flowers)

(3a) * het was (ondanks de luchtverversing door de tv-lampen) (snikheet)
(3b) het was (ondanks de luchtverversing) (door de tv-lampen snikheet)

These analyses differ with respect to the syntactic and thematic relations between the constituents (which themselves are identical in both analyses). Solving this type of ambiguity requires a semantic and pragmatic analysis; the parser must 'know' that one cannot use flowers to mow a lawn, and that TV lights produce heat rather than fresh air. Again, no system exists for this type of sentence analysis.

In order to bypass these parsing problems, we have attempted to derive the prosodic sentence structure directly, i.e. without exhaustive syntactic analysis. In other words, the prosodic domains (Phi and Int) are *not* established along the lines described above (viz. via an intermediate syntactic analysis). Instead, the PSS is derived directly from the orthographic input sentence, by rules which do *not* refer to a sentence's syntactic structure. Consequently, the resulting PSS can only approximate the theoretical prosodic structure, since not all relevant syntactic information is available for the prosodic analysis. Our approach is illustrated in Figure 3 below.

Figure 3: Alternative method for deriving the prosodic sentence structure (PSS) directly from an input sentence.



Some syntactic information, however, is of vital importance for a correct prosodic analysis. For example, the main verb (or verb group) in a sentence must be identified. This word (group) establishes a Predicate constituent, which should correspond to a separate Phi domain (Gee and Grosjean 1983). In Dutch, this Phi domain may separate the subject and object arguments of the predicate. Likewise, subordinate sentences must be identified, because they usually establish separate Int domains (see section 2). In section 3.2.1. below, we will argue that the information required for prosodic domain construction can often be derived from the syntactic word class.

As a first step, then, the words constituting the sentence must be provided with a syntactic label. Subsequently, the PSS is derived from both the orthographic input sentence (text string), and from the syntactic labeling of its constituent words. Finally, phrase boundaries and accents are derived from the PSS, as well as from the syntactic word labeling. A full listing of all rules (in quasi-SPE format) is given in Appendix 1.

3.2 from text to PSS

3.2.1 introduction

Two types of syntactic information are indispensable for prosodic sentence analysis. Firstly, lexical heads must be identified. This is achieved by identifying each word as either CW or FW. In theory, each last CW preceding an FW (or sentence boundary) constitutes a lexical/prosodic head. The CW-or-FW status of a word is determined by means of lexical lookup, in a lexicon which contains all Dutch FWs (554 entries). In addition, the lexicon contains another 300 CWs which behave anomalously for one reason or another. For all words, the syntactic word class is also specified. This syntactic labeling serves two purposes: it is used for (1) determining the syntactic labels of the words not found in the lexicon, and (2) the construction of prosodic domains, as explained below. See section 4.3. for more details on this lexicon.

Secondly, syntactic phrasing may be relevant for prosodic domains. Consequently, the syntactic phrasing must be known. In order to establish prosodic domains, however, an exhaustive syntactic analysis appears to be superfluous: it is not necessary to determine the structural relations between the words in a sentence. Returning to (1), for example, it is not necessary to decide which is the internal structure of the subject NP:

(1a) (Kasyapa's (great (war elephant)))
(1b) (Kasyapa's ((great war) elephant))
(1c) ((Kasyapa's (great war)) elephant)

Instead, it suffices to determine that these four CWs together establish a single constituent, which in turn establishes a separate Phi domain. This information can be derived from the syntactic class of the words in (1):

```
(1)    Kasyapa's great war elephant # turned aside ...
           N       A    N     N     #    V      A   ...
```

The Noun **elephant** and the inflected Verb **turned** cannot belong to the same syntactic constituent, as the Noun cannot be a specifier to an inflected Verb. Hence, **elephant** must be the lexical head of the constituent preceding the Verb **turned**; i.e., the two words must belong to separate syntactic constituents. Since both words are lexical heads of syntactic constituents, they cannot both belong to the same Phi domain. Consequently, a Phi boundary must separate these two words.

In more general terms, our approach is based on the fact that within a syntactic constituent, some possible sequences of syntactic labels are allowed, while others are not. If constraints on the possible label sequences are violated, then we may assume a syntactic boundary. Hence, syntactic constituency can be derived from the sequence of syntactic word labels, similar to the approach of O'Shaughnessy (1989). Not all syntactic boundaries, however, are equally important for the PSS. For the purposes of sentence prosody, it generally suffices to demarcate syntactic constituents which are respected by prosodic domains, while ignoring other syntactic structures.

3.2.2 word labeling

If an input word is found in the lexicon (mentioned in section 3.2.1. above), then syntactic label(s) is (are) copied from the lexicon. Words which are not found in this lexicon are given the prosodic label CW. Subsequent syntactic labeling concentrates on these CWs. During several tests, it was established that ca. 53% of the word tokens (in a large corpus of newspaper text) were found as FW in the lexicon.

Firstly, syntactic labels are generated on the basis of formal properties of the CW string. In the future, the syntactic label(s) for each word will be provided by a separate morphological parser (Baart and Heemskerk 1988; Heemskerk 1989), rather than by the following rules of thumb. At this moment, the generation of syntactic labels is triggered by e.g. affixes and orthographic conventions, as illustrated by the following rules (the symbol "|" indicates ambiguity; ":" indicates a sub-classification; "*" represents any character string):

(4) Undef → (VERB:INFL) / * **dt**
- het huis **brandt** (VERB)
  "the house burns"

(5) Undef → (UNDEF)|
(VERB:INFL) / * **{t,te,de}**
- hij **hoopte** te komen (VERB)
  "he hoped to come"
- de **hoogte** van de bergen (UNDEF)
  "the height of the mountains"

(6) Undef → (NOUN:PLUR)|
(VERB:(INFL|INF)) / * **en**
- de **bomen vangen** wind
  "the trees catch wind"

(7) Undef → ADV / * **{lijk,ig,zaam}**

Secondly, words with multiple syntactic labels (either words with labels from the lexicon, or CWs, with multiple labels generated by rule) must be disambiguated. Once again, this is done on the basis of restrictions on label sequences: linguistically motivated disambiguation can only be achieved when taking account of the context.

Two rules below illustrate this 'filtering' of the syntactic labels generated. Rule (8) employs a linguistic constraint on word sequences: an inflected Verb may not be preceded by an Article. Rule (9) employs a statistical constraint: Prepositions are usually followed by Nouns, rather than by Verbs. Such probabilistic observations are based on our analyses of large corpora of newspaper text.

(8) X|VERB → X / ART ___
- de **bakken** (NOUN)
  "the trays"

(9) NOUN| VERB → NOUN / PREP ___

- vogels broeden in **nesten** (NOUN)
  "birds breed in nests"
- hij wil zich daar in **mengen** (wrong: VERB)
  "he wants (to) himself there in mingle"

(10) NOUN| VERB → NOUN / POSS.PRON ___

- ze knipt mijn **haren** maandelijks (NOUN)
  "she cuts my hairs monthly"
- niemand kan mij **knippen** als zij (VERB)
  "nobody can me cut like she"

In addition, orthographic conventions guide the selection of syntactic labels: a word containing a hyphen is a compound, hence labeled as Noun; strings containing digits are labeled as Numeral; words starting with a capital (not sentence-initial) are proper names rather than Verbs; words longer than 13 characters are probably Nouns (e.g. **wapenhandelaren**) rather than long Verb strings (a Verb like **her-programmeren** is relatively rare), etc.

In the future, the disambiguation rules are to be refined and extended, since the morphological parser will generate a greater number of multiple syntactic labels, for a greater number of words, as compared to the rules of thumb described above.

3.2.3 prosodic domains

In section 3.2.1. above, we have explained that our prosodic domains are demarcated (i.e., prosodic boundaries are inserted in the sentence) on the basis of restrictions on the possible sequences of syntactic and prosodic word labels. Lower prosodic domains are strictly enclosed within higher domains: they cannot straddle the boundaries of higher domains ("strict layer hypothesis"; Nespor and Vogel 1986). In order to achieve this hierarchy, we start by demarcating the higher Int domains. Subsequently, Phi domains are demarcated within these Int domains. Note that this procedure deviates from the theoretical construction of prosodic domains (e.g. Nespor and Vogel 1982, 1986).

3.2.3.1 Int domains

Several rules demarcate Int domains by identifying subordinate clauses. As explained before, these rules insert an Int boundary between two adjacent words which cannot both belong to the same clause (as may be deduced from their syntactic labels).

At this point, it must be noted that our object language, Dutch, is an SOV language, where the inflected Verb takes the final position in subordinate clauses, and the second position in main clauses. English, by contrast, is an SVO language. Consequently, some generalizations in our rules are allowed in Dutch, while they do not apply to English.

Below follow some examples of rules inserting Int boundaries, on the basis of syntactic word labels. Rule (11) demarcates the end of a subordinate clause (with final Verb). Rule (12) is an example of a rule demarcating the beginning of a subordinate clause, while (13) demarcates two juxtaposed sentences; it is determined from the right-hand context of the Conjunctive word whether this word links two subclauses (and not two other constituents). The English examples only serve to illustrate the principle.

(11) 0 → IntBound / (VERB:INFL) ___ (VERB:INFL)

- omdat het geen haast **had ## deed** ik het later
  "because it no hurry had ## did I it later"
- the colours of the frescoes he **painted ## look** fresh and unfaded

(12) 0 → IntBound / (VERB:PART) ___ (NOT (VERB))

- hij is erin **geslaagd ##** om dit goed weer te geven
- he has **succeeded ##** in representing this correctly
- hij heeft **beloofd ##** morgen te komen

(13) 0 → IntBound / ___ (CONJ) {(VERB:INFL)}
{(FW)}

- hij snoof het stuifmeel **op ## en kreeg** direkt hooikoorts
- hij spreekt in losse woorden **## of in** een babytaaltje
- he picked up a stone **## but hesitated** to throw it away
- Thero excused himself **## and he** left the room

Other rules delimit Int domains on the basis of orthographic punctuation, such as comma's, quotes and parentheses. Some caution is required, however, because not all instances of these punctuation signs correspond to Int boundaries:

(14) is het geen kwestie van (taal)tolerantie om te kiezen voor "jij"?

In addition, there are also rules which demarcate complex ('heavy') sentence-initial constituents (mostly subject NPs).

3.2.3.2 Phi domains

For the demarcation of Phi domains, the same principle is applied as in the case of Int domains: insert a Phi boundary between two adjacent words which cannot both belong to the same Phi domain (see section 3.2.1.).

The most important of these rules uses the fact that the prosodic head of a Phi is always its rightmost lexical item. Usually, this head is a CW. FWs always belong to the prosodic head on their right-hand side, i.e., FWs belong to the right-hand Phi domain (see section 2.). Hence, a Phi boundary may be assumed between a CW (which is the prosodic head of the left-hand Phi domain) and a following FW (which is a specifier of the right-hand prosodic head):

(15) $0 \rightarrow$ PhiBound/ (CW) ___ (FW)
• she **prunes # the** red **roses # in** her garden

Of course, this requires additional rules for Phi domains whose prosodic head is not a CW, but rule (15) correctly identifies the majority of Phi domains.

Another rule aims at demarcating verbal clusters (i.e. predicates) as separate Phi domains, as in (16) below. This is motivated by (1) the fact that the verbal cluster may separate subject and object arguments, corresponding to separate prosodic domains, and (2) the special accentuation behaviour of verbal clusters (see section 3.3.2. below).

(16) only my sister # **wants to go skating #** in the summer

The above rules for Phi boundary insertion leave long strings of non-verbal CWs intact, i.e., such strings are not divided into multiple prosodic domains. Of course, this may be incorrect in cases where the CWs belong to separate thematic constituents, such as (17). Therefore, another rule (18) splits these CW strings, by inserting a prosodic boundary before an Adverb, which is usually not a specifier to its left-hand neighbour:

(17)	ik drink #	over	het	algemeen	liever	sterke	koffie
		FW	FW	N/CW	Adv/CW	Adj/CW	N/CW

(18) $0 \rightarrow$ PhiBound/ ___ (ADV)
• ik drink # over het algemeen # **liever** sterke koffie

Finally, some of the resulting Phi domains (e.g., those containing only an FW prosodic head) are merged with their left-hand neighbour.

3.3 from PSS to prosody

In the second part of our algorithm, abstract prosodic phenomena are derived from the PSS (constructed by the first part, described in section 3.2. above). At this moment, the PSS is used only to determine phrase boundaries and accentuation, as mentioned in section 1. In theory, however, it is also possible to derive other aspects of the output sentence prosody from the PSS, such as phonological sandhi phenomena.

3.3.1 phrasing

At this moment, the PSS is used directly to split the input sentence into separate phrases. Each resulting Int boundary becomes manifest as a prosodic break, which is realised in the output synthetic speech as a pause (250 ms) accompanied by an appropriate F_0 movement. This procedure, however, results in disfluency in the speech output, thus inhibiting (rather than facilitating) correct perception of the synthetic speech. Apparently, too many breaks are present in the speech stream, and the phonetic means by which they are realised may be too strong. To improve this situation, two solutions are being investigated in a follow-up research project (SPIN/ASSP *PROS2*, by Arthur Dirksen).

Firstly, the *number* of Int boundaries can be reduced, by means of the restructuring of Int domains (as mentioned in section 2.). In other words, two adjacent Int domains are collapsed into a single one, by rules deleting the intermediate Int boundary. Like other tasks performed by our algorithm, this restructuring should be guided by theoretical considerations:

--obligatory Int boundaries (e.g. those based on orthographic punctuation) must be maintained;

--shorter Int domains (in terms of number of words and syllables) are more prone to collapsing;

--the resulting Int domains should be of approximately equal length;

--restructuring depends (to a certain extent) on the syntactic functions of the Int domains involved; a possible hierarchy could be the following (boundaries ordered from 'heavy' to 'light'):

[1] boundaries resulting from orthographic punctuation (non-restrictive relative clauses, appositions, etc.);

[2] boundaries following a heavy sentence-initial constituent (introductory sub-clauses, complex NPs, etc.);

[3] boundaries preceding a sub-clause which functions as sentence modifier (often beginning with **omdat, om te**);

[4] boundaries preceding a sub-clause which functions as argument to the Verb (often beginning with **dat**);

[5] boundaries preceding a restrictive relative clause (often beginning with **die**);

[6] boundaries preceding an extraposed constituent (often a PP).

Secondly, the *phonetic realisation* of Int boundaries as prosodic breaks in the output speech could be differentiated, depending on the 'weight' of the Int boundary, as well as on the length and function of the corresponding Int domains. Relatively weak Int boundaries may be realised by phonetic means which are perceptually less salient, e.g. prepausal lengthening rather than F_0 movements.

3.3.2 accentuation

A fitting intonation contour of a sentence can be derived automatically (at least, in the case of Dutch), if phrase boundaries and accents are known ('t Hart and Collier 1975; Van Wijk and Kempen 1985). In this section, we will describe how accents can be derived from the PSS. The result is an abstract word property: either plus or minus *accent*. This property becomes manifest primarily by intonational means, but also in segmental durations, intensity, vowel reduction, etc. (Nooteboom and Kruyt 1987; see also section 1).

The accentuation component of our algorithm attempts to imitate several theoretical aspects which are known to affect accentuation. These factors will be discussed first; subsequently, our approach in accentuation is discussed.

3.3.2.1 theory

Firstly, FWs seldom receive accent. It is a consequence of their FW status that they have a reduced prominence, and are usually left unaccented (Kruyt 1985).

Secondly, in Dutch, the prosodic head (i.e., last CW in Phi domain) is the word which receives *integrative accent*. An accent on this prosodic head lends prominence to the domain as a whole, and not only to the 'carrier' word (Baart 1987).

Thirdly, discourse context, specifically the distinction between 'given' and 'new' information, strongly influences accentuation (Fuchs 1984). A domain must be unaccented if it refers to information which is (supposed to be) already 'given' to the listener ('known', 'old'). Usually, only domains introducing 'new' information are accented (by means of an integrative accent), as demonstrated below:

(19a) he came # by **CAR**
(19b) his **car** # was BLUE

Kruyt (1985) and Terken (1985) have confirmed the role of this distinction between 'given' and 'new' in accentuation. Apparently, listeners rely on this distinction for their understanding of the spoken utterance (Terken and Nooteboom 1987).

Fourthly, the thematic relations between prosodic domains play an important role (Gussenhoven 1984; Baart 1987). As an example, consider the accentuation of the predicate (i.e., prosodic domain corresponding to the main verb or verbal group). Even if it refers to new information, this word (group) is usually not accented (20). Under two conditions, however, the predicate must be accented.

--if the predicate is not adjacent to (any of) its argument(s); this may happen if a non-argument constituent (sentence modifier, adverbial phrase) intervenes between the predicate and its arguments, as in (21).
The relevance of the modifier status of the intervenient can be inferred from comparing **met de bloemen** (20b) and **met de machine** (21). Neither are arguments to the predicate, but only the latter one is a sentence modifier, while the former one depends syntactically on **het gazon** (yielding a complex argument **het gazon met de bloemen**).

--if all of the arguments of the predicate are unaccented; this may happen if the arguments convey given information, or if they contain only FWs, as in (22).

(20a) ik heb # het GAZON # gemaaid
(I've # the lawn # mown)
(20b) ik heb # het gazon # met de BLOEMEN # gemaaid
(I've # the lawn # with the flowers # mown)

(21) ik heb # het GAZON # met de MACHINE # GEMAAID
(I've # the lawn # with the machine # mown)

(22)	(hoe is het met het gazon?)	(how is the lawn?)
	ik heb # het gazon # GEMAAID	(I've # the lawn # mown)
	ik heb # het # GEMAAID	(I have # it # mown)

Finally, rhythmic factors influence accentuation. The occurrence of multiple adjacent accents is avoided. In such cases, one of the accents is removed (23b) or shifted to a different word (24b) (see Kager and Visch 1988, Visch 1989 for a lengthier discussion).

(23a)	de HEFTIG PROTESTERENDE BUREN
(23b)	de HEFTIG protesterende BUREN
	(the fiercely protesting neighbours)
(24a)	hij heeft # de hele NACHT # GELEZEN
(24b)	hij heeft # de HELE nacht # GELEZEN
	(he has # the whole night # read)

3.3.2.2 algorithm

Our accentuation rules attempt to imitate the theoretical account of accentuation discussed above. In addition, the rules refer to the PSS (section 3.2.3.), as well as to the syntactic word labeling (section 3.2.2.).

Firstly, CWs are accented. Two types of words are excluded from this accentuation, viz. *verbs* and semantically 'empty' words (e.g. **gulden** "guilder"). In addition, some idiosyncratic words are also accented (e.g. **nooit** "never"). The two types of anomalous words are listed in the lexicon.

Obviously, this procedure yields too many accents. Subsequently, two types of rules strip CWs of their accent, under conditions where an accent is known to be wrong: [1] rhythmic deaccentuation, [2] deaccentuation of words conveying 'given' information. In addition, [3] Verbs (or verb groups) are accented in some specific contexts.

3.3.2.2.1 rhythmic deaccentuation

In order to obtain rhythmic accentuation patterns, the middle one of three adjacent accented CWs is de-accented. Both the PSS and the syntactic word labeling are taken into account: all three words must belong to a single Phi domain, and they must belong to certain sequences of syntactic categories:

(25)	*QuantNum*	*X*	*Noun*	
	drie	duizend	boeken	(three thousand books)
	zeer	lage	temperatuur	(very low temperature)
(26)	*Adverb*	*Adj*	*Noun*	
	evenwijdig	lopende	spoorlijnen	(parallel running tracks)
	zeer	verbaasde	toeschouwers	(very amazed spectators)

Note that these conditions are stricter than one would expect on the basis of (23, 24), in order to avoid incorrect over-applications. In the future, more contexts could be added to trigger rhythmic deaccentuation.

3.3.2.2.2 deaccentuation of 'given' information

The second type of rule de-accentuates words which can safely be assumed to convey given information. Although the scope (window) of our algorithm is limited to a single sentence of the input text, the rules can nevertheless infer whether words in the sentence under analysis have occurred before. Deictic qualifiers (such as **dit**, **deze** "this", **dergelijke**, **zulke** "such", etc.) imply that the following term conveys information which has already been introduced ('given'). Any words following this cue word in the same Phi domain are de-accented, as indicated by (27):

(27a) he came # by CAR
(27b) he has BORROWED # **this vehicle** # from a FRIEND

Two groups of deaccentuating ('known') qualifiers are distinguished, for reasons related to verb accentuation. Both the first (28) and the second group (29) trigger de-accentuation of subsequent words within the Phi domain in which they occur. Note that the second group contains all words ending in **-ere**, i.e., all inflected forms of Adjectives in comparative. Obviously, the basis of this rule type (as well as of the inclusion of ***ere** words in the set of deaccentuating qualifiers) is probabilistic. Errors occur in a minority of cases, but examples like (30) certainly represent the majority:

(28) die *(if Pronoun)*, dat *(if Pronoun)*, dit, deze, dergelijk(e), zulk(e), menig(e), meeste, zo'n *(unless followed by Numeral)*

(29) ander(e), huidig(e), volgend(e), vorig(e), overige, evenmin, beide, elders, dezelfde, diezelfde, eenzelfde, zelfde, *ere

(30a) het GEVOLG # van **zulke** temperaturen # is VRESELIJK
(30b) ONDERZOEK # heeft # de **hogere** sterfkans # BEWEZEN

In addition, de-accentuation of 'given' information is applied in a few specific contexts: epitheta before a proper name, sentences following (in)direct speech quotation (in newspaper text, these 'trailing' sentences usually start with **zo** or **aldus** "according to"), and unit measures following a numeral. The examples below illustrate these rules.

(31) **queen** BEATRIX # is the SUCCESSOR # of **princess** JULIANA

(32) de DIEF # heeft BEKEND ## **aldus** een **woordvoerder**
(the thief # has confessed ## according to a spokesman)

(33) UTRECHT # lies # FOURTY **kilometers** # from AMSTERDAM

3.3.2.2.3 verb accentuation

Under certain conditions, the predicate (verb group) must be accented, as discussed in section 3.3.2.1. above. This is required if a sentence modifier intervenes between the verb (group) and its arguments, or if all of the arguments are unaccented (or [-focus], in the words of Gussenhoven 1984 and Baart 1987) -- or simply absent.

The rules which accentuate a Verb work on single CW Verbs, as well as on longer sequences of Verbs, of which the first CW is accented. In addition, the rules work on complex Verbs, of which the stem and particle can occur separately. Stem and particle occur together only in participles (34a)[2]. But in infinitive constructions, the infinitive marker **te** can be interposed (34b), whereas in inflected forms in Dutch, the inflected stem [in second position in main clause] may be far away from the particle [in the 'original' clause-final position], as in (34c). An interesting property of these separable verbs is that the *particle* is accented, even if the stem is far away (34b,34c):

(34a) ik **heb** # mijn MOEDER # VANDAAG # **OPGEBELD**
(I've # my mother # today # phoned)
(34b) ik heb GEPROBEERD # mijn MOEDER # VANDAAG # **OP te bellen**
(I've tried # my mother # today # to phone)
(34c) ik **belde** # haar # VANDAAG # **OP**
(I phoned # her # today # -)

The accentuation of these separable Verbs is identical to other Verbs, and incorrect non-accentuation of the *particle* part of separable Verbs is accordingly a serious error. However, particles are often identical to Prepositions (e.g. the ambiguous **op** in (34)). This presents difficulties in detecting verbal particles. Some can be identified because they occur immediately before the infinitive marker **te** (34b); others may be identified because they occur in clause-final position (34c), hence cannot introduce a PP, and hence cannot be Prepositions. The rules to be discussed below treat these 'stranded' clause-final particles as Verbs, so that they can carry the accent of separable Verbs.

The first rule accentuates the verb (group) after an Adverb (35) or adverbial phrase. The verb (group) must be followed by an Int (or sentence) boundary. This serves to guarantee that the verb is syntactically clause-final and not clause-medial, where it would typically be followed by (and hence, be adjacent to) an argument. The rule can detect adverbial phrases consisting of PPs through their initial Preposition. That is, PPs starting with e.g. **ondanks, sinds**[3] are used as sentence modifiers (36a), rather than as constituents within an NP. In addition, 'locative' PPs (**in** followed by a proper name) also qualify as sentence modifiers (36b).

(35) hij heeft # het GAZON # **VANDAAG$_{\mathbf{Adv}}$** # **GEMAAID**
(he has # the lawn # today # mown)

---

2 Dutch orthography requires that the two parts of the separable verb are written as a single word in these cases.

3 'Adverbial' Prepositions are : **door** ("through, by"), **met** ("with"), **na** ("after"), **ondanks** ("in spite of"), **per** ("per"), **sinds** ("since"), **tijdens** ("during"), **via** ("via"), **volgens** ("according to"), **wegens** ("because of"), **zonder** ("without").

(36a) hij heeft # het GAZON # **ondanks** de REGEN # **GEMAAID**
(he has # the lawn # in spite of the rain # mown)

(36b) de SPION # werd # **in** ROTTERDAM # **GEARRESTEERD**
(the spy # was # in Rotterdam # arrested)

If the predicate is accompanied by an un-accented argument, then the predicate (verb group) must be accented. Since the deictic specifiers (28) indicate un-accented terms, we can employ these cues to arrive at a more adequate accentuation of CW Verbs, as in (37). However, only one subset (28) of deaccentuating specifiers triggers Verb accentuation. We have found that the other type of deaccentuating specifiers (29) frequently takes the integrative accent in an argument + predicate construction, as in (38).

(37) de ZEEHONDEN # hebben # **deze** lage temperaturen # **OVERLEEFD**
(the seals # have # these low temperatures # survived)

(38) ze heeft # de **VOLGENDE** trein # **genomen**
(she has # the next train # taken)

Likewise, the verb (group) is accented if its (object) argument is a Pronoun, as in (39) (Pronouns are un-accented because they are FW, and because the Pronoun usually refers to a term which has already been introduced). Finally, the verb (group) is accented if there is no object argument; this can be assessed from the fact that the verb is preceded by a single Phi domain (40a) which does not contain any Pronouns (cf. (40b)).

(39) de ZEEHONDEN # hebben # **het** # **OVERLEEFD**
(the seals # have # it # survived)

(40a) de ZEEHONDEN # zijn **GESTORVEN**
(the seals # have died)

(40b) dat **ik mijn** ZUSTER # heb **ontmoet**
(that I my sister # have met)

4. IMPLEMENTATION

4.1 introduction

The program PROS generates accentuation and phrasing for Dutch orthographic text, via the prosodic sentence structure. From a functional viewpoint, this means that a sentence from the input text is provided with two types of markers:
(a) accents: which words are to be accented in a spoken version of the sentence;
(b) phrases: the boundary positions between major prosodic phrases.

The input of PROS consists of a string of characters (graphemes), which can be read either from the terminal keyboard or from an external text file. The input text is automatically divided into sentence units. The output consists of a string of characters (graphemes), enriched with accentuation and phrasing markers. Words carrying accent are put out (by default) in uppercase characters; prosodic boundary markers are inserted between major phrases in the output grapheme sequence. Output is written both to the terminal screen and to an external text file.

PROS is written in VAX-Pascal (v3.4 and higher), running on a VAX 11/750 or microVAX 2000 under VAX/VMS (v4.3 and higher). Since the program modules contain constructs which are specific to the (non-standard) VAX-dialect of Pascal, minor adjustments are necessary in order to run PROS on other systems. Total size of the source code (including comments) is 485 kByte; the executable code amounts to 116 kByte (excluding the lexicon, which is to be stored in on-line memory during run-time).

4.2 design

The primary task of the program PROS is performed by (routines declared in) the actual analysis modules. These analysis rules are described in more detail in section 6.5. below. The analysis routines require additional auxiliary routines, which take care of tasks such as opening and closing files, reading and writing sentences, initializing variables, etc.

The analysis modules, however, are also used by other programs which use the same rules for deriving sentence prosody [these programs are described in chapter 6]. Consequently, two types of auxiliary routines are to be discriminated: (1) those necessary for the prosodic analysis routines, and (2) other program-specific (auxiliary) routines. The former routines are declared at a higher level (viz. in the global environment) as compared to the latter routines (viz. in the main program).

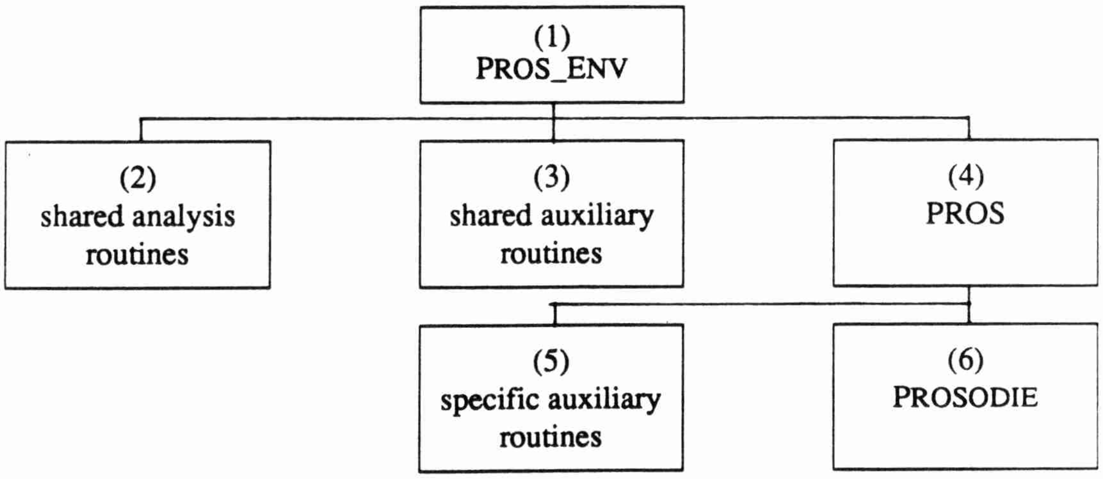
As a further complication, the analysis routines are *not* called directly from within the main program PROS. Instead, a (globally declared) routine PROSODIE is called. The declaration of this routine PROSODIE in a separate module makes it possible to modify the flow of prosodic analysis in an easy way, without altering (and re-compiling) the main program itself (and its dependent modules).

In summary, the modules constituting the program PROS are grouped as follows:

(1) global environment (global CONST, TYPE, VAR and routine declarations, shared with auxiliary programs):
--PROS_ENV
(2) modules with prosodic analysis routines (shared with auxiliary programs):
--LABEL_WORDS, ADJ_FWLABELS, ADJ_VERBLABELS, ADJ_LABELS,
--INSERT_BOUNDS, ADJ_BOUNDS,
--SELECT_ACCENTS, SELECT_VERBACCS, ADJ_ACCENTS;
(3) modules with auxiliary routines (shared with auxiliary programs):
--ALGROUTS, LEXTRIEROUTS, POINTROUTS, STRINGROUTS,
--LEESSCHRIJF[4];
(4) main program (global CONST, TYPE, VAR and routine declarations, specific for each program; implementation of main program):
--PROS (or other stand-alone program from the PROS family);
(5) modules with auxiliary routines (specific for each program):
--GLOBALROUTS, INITIALIZE, OPTIONS;
(6) module containing routine PROSODIE (specific for each program):
--PROSODIE.

The internal dependence between these modules is controlled by the 'inheriting' of information in higher-level modules (by means of the VAX-Pascal attributes ENVIRONMENT and INHERIT). The dependence relations between the above six groups of modules are illustrated in Figure 4 below.

Figure 4: Dependence relations (as specified by VAX-Pascal directives "ENVIRONMENT" and "INHERIT") between six groups of modules, together constituting the program PROS for prosodic sentence analysis, phrasing and accentuation.



[4] Module LEESSCHRIJF is not used by the auxiliary program PROS_DS (discussed in section 6.1).

The module groups (1-3) (see above) are to be found in a logical directory **usr$prosenv**, which must have been defined before compiling and linking any of the modules.

4.3 the lexicon

4.3.1 word forms

The lexicon **ProsLex** (see Appendix 2.2. for a full listing) was originally derived from the Dutch ULEX lexicon (Van den Broecke, Aerts, Reizevoort, Veenhof, Lammens and Elstrodt 1987). As a first approximation, words which qualified as FW (considering their syntactic label in ULEX) were selected. Thus, the lexicon contained all Dutch Articles (N=5), Prepositions (N=60), Pronouns (N=80), Conjunctives and Complementizers (pooled N=48). For these 193 word types, Van den Broecke et al. (1987) report a pooled token frequency of 36% in newspaper texts. This first lexicon was modified in several ways:

--Adverbial FWs were added to the lexicon [e.g. **daar, er** "there"].

--In Dutch, new Complementizers [**waar**+Prep] and Adverbs [**daar** + Prep, **er** + Prep, **hier** + Prep] can be constructed. Not all combinations of the "root" with a Preposition were included in the ULEX lexicon. If such a word occurred regularly in the newspaper text corpus, then it was included in the ProsLex lexicon.

--For verbal FWs, the infinitive, participle, and all inflected (past and present) forms were added to the lexicon. A total of 31 Verbs[5] were considered as semantically empty (un-accentable), and hence as FWs.

--Content words (CWs) were included in the lexicon (with their appropriate syntactic label), if they behave anomalously with regard to phrasing and/or accentuation (e.g. because they are part of an idiomatic expression in which no prosodic boundary may be interposed, or because the CW may not be accented).

--Particles of separable Verbs (e.g. **toe** as in **toe+zenden**) were stored with a new syntactic label (*Satellite*), unless the particle was already stored as a Preposition (e.g. **op** as in **op+bellen**).

--For some words, the syntactic label was adjusted, in order to arrive at a labeling which was more relevant for our purposes. For example, the label of **beide** "both" was changed from Pronoun to Numeral, since this word usually functions as a numeric quantifier in NPs.

4.3.2 lexical information

As has been explained before, each entry (word form) in the lexicon is provided with [1] a syntactic label (and for some labels, [2] an additional sub-label). Moreover, [3] the FW or CW status ("prosodic label") of a word is indicated by means of

[5] **blijken, blijven, doen, gaan, geven, gooien, halen, hebben, hoeven, houden, komen, krijgen, kunnen, laten, leggen, liggen, lijken, maken, moeten, mogen, nemen, staan, vinden, vragen, werpen, willen, worden, zetten, zijn, zitten, zullen.**

another label. This was necessary because there is no 1:1 relation between syntactic and prosodic labels: Verbs and Adverbs can be found in both CW and FW categories. Finally, it is indicated [4] whether accentuation of a word is either free (as for most CWs), obligatory (as for emphatic words, e.g. **altijd** "always") or forbidden (as for most FWs, and some CWs). Ambiguous words are signalled by a double entry with different labels. All labels are coded as digits in the lexicon, which are reconverted by PROS into 'enumerated type' labels, according to the coding scheme below.

Table 1: Coding scheme for labeling in PROS lexicon.

label	value	meaning		sub	value	meaning
[1]	0	VERB	>	[2]	0	(undefined)
Synt	1	NUMeral		SbV	1	PV ("inflected")
	2	SATellite			2	PARTiciple
	3	COMPlementizer			3	INFinitive
	4	ARTicle				
	5	PREPosition				
	6	PRONoun				
	7	CONJunctive				
	8	ADVerb				
	9	(undefined)	>	[2]	0	(undefined)
				SbUnd	1	NOUN
					2	ADJective
[3]	0	Content Word				
Pros	1	Function Word				
[4]	0	(free)				
Acc	1	+accent ("obligatory" accent)				
	2	-accent ("prohibited" accent)				
[5]	0	(not used)				

In addition to these four labels, additional lexical features may be specified for an entry in the lexicon[6]. These features function as "triggers" to specific analysis rules in PROS. Thus, the rules themselves do not refer to lexical items (word strings), but only to lexical features. For example, an analysis rule which refers to personal pronouns in its context does not enumerate all these pronouns. It is merely verified whether the word in question is provided with the appropriate lexical feature ^X, with which all personal pronouns in the lexicon are specified.

The lexical features and their "meaning" are listed in Appendix 2.1; the PROS lexicon itself (word form strings, 4 label digits, and lexical features) is given in Appendix 2.2.

[6] Lexical features are indicated in this report with a caret (^): **^61** = feature number 61.

4.4 data structures

4.4.1 sentence: double-linked list

The program PROS processes orthographic sentences. These sentences must be available on-line for inspection and manipulation. Each word of the input sentence is stored in a separate word record; word records are interconnected into a dynamic "double-linked list" by means of pointers. Hence, the 'scope' or 'window' which PROS has on the input text is only one sentence. Relations (between words) across sentences cannot be established, although they may be relevant for accentuation and phrasing (as explained above). In module PROS_ENV, word records and pointers are declared with the following (variant) fields:

```
TYPE
Wijzer_Type   = ^WordRec_Type
WordRec_Type = RECORD
                     Word     :word string, max 30 chars
                     Acc      :boolean, word accented
                     Synt     :syntactic class label
                           SubVerb : synt sublabel
                           SubUndef: synt sublabel
                     Pros     :prosodic class label
                     LexFeats:array of feature digits
                     Ambigu   :boolean, word ambiguous
                           Alt1 : features option 1
                           Alt2 : features option 2
                     Prevw    :pointer to previous word record
                     Nextw    :pointer to next     word record
                     END;
VAR
First, Current, Last : Wijzer_Type ;
```

Upon reading a sentence from the input text, new pointers are created and appended following the last record in the double-linked list. Punctuation marks (period, quotes, comma, hyphen, parentheses, question mark, space) are interpreted as word delimiters, and usually treated as words (period, exclamation mark and space are never treated as words; hyphens and quotes are treated as a word only under special conditions). During prosodic analysis, new records (with prosodic boundaries as word strings) are inserted between word records in the double-linked list. After prosodic sentence analysis, the output sentence (content of the double-linked list) is written to the output devices. Finally, the existing double-linked list is deleted, and a new loop begins.

4.4.2 lexicon: trie

Upon initialisation, the external lexicon is read from a text file (with logical name **ProsLex**) into a memory-resident trie structure (Knuth 1973). This structure consumes relatively little memory, and allows fast retrieval of stored words. Each record contains one character, as well as pointers to adjacent records. The actual storing and searching routines are adopted from Lammens (1989). Figure 5 below illustrates this trie structure. Note that six words (totalling 33 characters) can be stored in 17 one-character nodes. With larger lexicons, the reduction in memory requirement is even greater.

Figure 5: Schematic representation of a lexicon with trie structure, containing the words "bel, beland, beul, circa, circuit, circus" are stored. Boldface characters indicate that a word terminates with that character.

```
[root]---b---e---l---a---n---d
         |       |
         |       u---l
         |
         c---i---r---c---a
                         |
                         u---i---t
                             |
                             s
```

In module PROS_ENV, the records in the lexicon trie are declared with the following fields:

```
LexNodeWijzer_Type= ^LexNodeRec_Type
LexNodeRec_Type    = RECORD
                          Character :character in node
                          Terminal  :boolean, end word
                          Yes       :pointer to next char
                          No        :pointer to alt  char
                          LexFeats  :array of feature digits
                          Ambigu    :boolean, ambiguous
                               Alt1: features option 1
                               Alt2: features option 2
                        END
```

4.5 auxiliary modules

4.5.1 general routines

function **JaNee**
(Prompt:LongString_Type; Default:BOOLEAN)
: BOOLEAN;
Asks the user a Yes/No question, with *prompt* as prompting text and *default* as default response; response "y(es)" returns TRUE, "n(o)" returns FALSE.

function **LeesDigit**
(Prompt:LongString_Type; Default:Digit_Type)
: Digit_Type;
Asks the user for a digit response 0..9, with *prompt* as prompting text. This digit may be an option number from a menu. The (default) response is the return value.

procedure **Inq_Open_File**
(VAR inqfile:TEXT; nwf:BOOLEAN;
prstring:LongString_Type; dfstring:LongString_Type) ;

Inquires for RMS file name, with *prstring* as prompting text. Open logical file *inqfile* with this RMS name, either as a new (write) file or as an old (read) file, depending on *nwf*. *Dfstring* contains the default RMS file specification.

function **ExistWord**
(search:Digit_Type; neighbour:BOOLEAN;
SyntLab:SyntMark_Type; ProsLab:ProsMark_Type)
: BOOLEAN ;

Verifies whether a word matching the partial word specification (i.e., a word with fields Synt = *syntlab* and Pros = *proslab*) exists, either preceding (*search*=1) or following (*search*=2) the parameter word *x*^ in the sentence. Parameter *neighbour* indicates whether the word must be a direct neighbour of the parameter word. *Syntlab* and *proslab* are tested in conjunction, i.e., both must match for the same word. The parameter values *Syntlab*=Undef and *proslab*=NN match with every word specification for Synt and Pros, respectively.

function **NrSubseqWords**
(SyntLab:SyntMark_Type; ProsLab:ProsMark_Type; x:Wijzer_Type)
: PosInt ;

Investigates how many subsequent word records, with labels *syntlab* and *proslab*, can be found from parameter pointer *x* onwards (including x^). Syntlab=UNDEF and ProsLab = NN match any field in the records). This function returns a positive integer.

function **Known_Constituent**
(VAR x:Wijzer_Type; search:Digit_Type)
: BOOLEAN ;

Decides whether the constituent (word record list) containing the parameter pointer *x* (or the adjacent constituent in the *search* direction) qualifies as a "known" constituent, i.e., starting with a word introducing "known", "given" or "old" information.
If the function result is TRUE, then *x* is set to as to point to the "Known-Qualifier" word, otherwise it is set to NIL. Search direction: 1 = LEFT/Back, 2 = RIGHT/Forward.

function **UnSpecified**
(z : LexFeats_Type)
: BOOLEAN ;

Verifies whether cells 1..5 (corresponding with fields Synt, Pros, SubVerb-or-SubUndef, Accent) of the feature array *z* are un-modified (with default values).

function **Cvt_Feat_Synt**
  (x : LexFeats_Type)
  : SyntMark_Type ;
    Converts the digit code in cell [1] of array *x* to syntactic code (enumerated type).

function **Cvt_Feat_SubVerb**
  (x : LexFeats_Type)
  : VerbSub_Type ;
    Converts the digit code in cell [2] of array *x* to sub-verb syntactic code.

function **Cvt_Feat_SubUndef**
  (x : LexFeats_Type)
  : UndefSub_Type ;
    Converts the digit code in cell [2] of array *x* to sub-undef syntactic code.

function **Cvt_Feat_Pros**
  (x : LexFeats_Type)
  : ProsMark_Type ;
    Converts the digit code in cell [3] of array *x* to prosodic code.

function **FeatValue**
  (x:LexFeats_Type; N·FeatNr_Type)
  : BOOLEAN ;
    Verifies whether the feature array *X* contains a non-zero value in cell number *n*, i.e., whether the array of features contains a positive flag for the lexical feature with number *n*.

procedure **SelectVariant**
  (VAR x:WordRec_Type; s:SyntMark_Type) ;
    For an ambiguous word (passed as parameter *x*), this routine selects one of the two feature variants (Alt1 or Alt2), viz. the one corresponding with field Synt=*s*. Thus, the word is set to *s* if this syntactic label is one of its possible ambiguities. If syntactic label *s* is not present in either one of the ambiguities, nothing happens.

procedure **Message**
  (nr:errnr_type; parstr:LongString_Type) ;
    Displays a Warning or Error message which corresponds to the parameter value *nr*. For some messages, the parameter string *parstr* is inserted in the output message (e.g. containing the current focus word).

procedure **Init_AfkList** ;
    Initializes the global array *AfkList* with the following highly frequent Dutch abbreviations, together with their solutions: "bijv, bv, dr, drs, hr, ing, ir, mej,

mevr, mw, prof, enz, etc, ma, di, din, wo, woe, do, don, vr, vrij, za, zat, zon[7], mr, tel, st".

procedure **Define_ScreenOutPut** ;

Asks user for a value of global variable *screenmode*, which controls screen output of the analyzed sentences. Possible values are 1 [no screen output], 2 [sentence word strings only, no labels], 2 [word strings and labels], 4 [proportion of input file which has been processed]. Value 4 is not allowed if PROS works in interactive mode (DataFile=**sys$input**).

procedure **Define_Monitor_Status;**

Sets up monitor breakpoints (watchpoints) in the derivational process, by calling global routine SetMonitor, and asks user for a value of global variable *moniall*, which controls whether non-applications are also to be monitored.

procedure **Initialize;**

Sets global counters to zero; initializes breakpoints (module names); initializes abbreviations list (by calling routine Init_AfkList); defines breakpoints (by calling routine Define_Monitor_Status; defines global variable *applall* (which controls whether non-applications of each rule must also be registered); opens input and output files; defines screen and file output formats.

4.5.2 string routines

function **Centre**
(String:LongString_Type; Letter:CHAR)
: LongString_Type ;

Returns a text line (80 chars) with *string* in the centre, surrounded by *letter*.

function **Vowel**
(ch:CHAR)
: BOOLEAN;

Verifies whether parameter character *ch* corresponds to any of the vowel phonemes, i.e., whether *ch* belongs to the character set [A,E,I,O,U,Y, a,e,i,o,u,y].

function **Cons**
(ch:CHAR)
: BOOLEAN;

Verifies whether parameter character *ch* corresponds to any of the consonant phonemes, i.e., whether *ch* belongs to the character set [A..Z, a..z] - [A,E,I,O,U,Y, a,e,i,o,u,y].

[7] The abbreviation **zo** for **zondag** was removed from this list, since this string occurs frequently as a normal word at the end of a sentence (followed by a period, which suggests erroneously that zo is used as an abbreviation).

function **SameWord**
(word1,word2: WordString_Type)
: BOOLEAN;
Verifies whether the two parameter word strings words are identical (ignoring case differences).

function **Hoofdletter**
(word:WordString_Type)
: BOOLEAN ;
Verifies whether parameter string *word* starts with a capital (uppercase char), e.g. in case of proper names.

function **CapString**
(VAR word:WordString_Type)
: WordString_Type;
Returns a word string, which is the uppercase equivalent of the parameter string *word*. This is used to indicate word accentuation.

function **DelBlanks**
(y:LongString_Type)
: LongString_Type ;
Returns the input text line *y*, in which all blanks are deleted.

function **OffSet**
(word,target: WordString_Type)
: BOOLEAN;
Verifies whether string *word* contains string *target* as its offset, i.e., whether *word* ends with *target*; this routine is used to detect inflectional suffixes in word strings.

function **Onset**
(word,target: WordString_Type)
: BOOLEAN;
Verifies whether string *word* contains string *target* at its onset, i.e., whether *word* starts with *target*; this routine is used to detect prefixes in word strings.

function **WordListMember**
(w:WordString_Type; L:WordList_Type)
: BOOLEAN ;
Verifies whether word string *w* occurs in the array *L* of word strings.

4.5.3 pointer routines

function **Walk**
(VAR x:Wijzer_Type)
: BOOLEAN;

Sets parameter pointer *x* to its righthand neighbour, viz. the next record in the list (if there is a next one); returns FALSE if *x* now equals NIL, otherwise returns TRUE.

procedure **Append_Rec**
(VAR x:Wijzer_Type) ;

Adds a new record at the end of the linked list, viz. **after** the pointer parameter *x*, and sets default field values for this new record. The new record is **not** linked with any records following *x*.

procedure **Insert_Rec**
(VAR f,c:Wijzer_Type) ;

Inserts a new record in the linked list (starting with *f*) **before** the parameter pointer *c*, sets variable *c* to this new record, and sets default field values for this new record.

procedure **Delete_Rec**
(VAR f,c:Wijzer_Type) ;

Deletes the record to which *c* points from the linked list (starting with *f*), and sets pointer *c* to the record preceding the deleted one.

procedure **Reverse_Recs**
(VAR f,c,l:Wijzer_Type) ;

Reverses the relative position of records (A) *C*^.prevw and (B) *C*, unless *C*^ has no predecessor. The output parameter is a pointer to the identical **position** in the list, now pointing to a record containing data A instead of B. Parameter *F* points to the first record in the double-linked list, *L* points to the last one.

procedure **Delete_List**
(VAR x:Wijzer_Type) ;

Deletes all records in the sentence (double linked list) from *x*^ onwards, including parameter pointer *x*, and dispose of pointers (make memory free).

4.5.4 lexicon routines

procedure **Init_LexTrie** ;

Builds a trie structure, reads word strings and word label digits from lexicon (text) file, and stores both in trie. This procedure results in a memory-resident lexicon trie, with characters as nodes, and global variable *LexRoot* as root pointer. Number of words, ambiguous words, characters and nodes are reported.

procedure **Store**
(VAR word:WordString_Type; features:LexFeats_Type) ;
Stores string *word* with lexical *features* in the lexicon trie. This routine results in a lexicon trie structure in which *word* has been added as a path, with its word labels stored in the last record of that path. Called by routine Init_LexTrie.

procedure **Sub_Store**
(VAR LexNode:LexNodeWijzer_Type) ;
Stores one character of *word* (from routine Store) in lexicon trie, adjusts counters. If *word* is now fully stored (end-of-word reached), then its *features* are stored in the current (terminal) trie node [by calling routine Translate]. Otherwise, the next character of *word* is stored, by calling calling Sub_Store recursively, with the appropriate [matching or non-matching] next LexNode as parameter.

procedure **Translate**
(ft:LexFeats_Type; VAR x:LexNodeRec_Type) ;
Stores the feature array *ft* in trie node *x* (last node of word path). For non-ambiguous words, features are stored in field LexFeats; if this field is already specified, then the current features specify an ambiguity: field Ambigu is set to TRUE, field LexFeats is copied to Alt1, *ft* is stored as Alt2. Called by routine Sub_Store.

function **ZoekLex**
(VAR InWord_Rec:WordRec_Type)
: BOOLEAN;
Searches for the word string from parameter record *inword_rec* in the lexicon trie; if the word string corresponds to a path in the trie (i.e. word found), then the additional word features are copied to *inword_rec* (fields Synt, Pros, Ambigu, Accent, LexFeats, (Alt1, Alt2)). The return value indicates whether the search has succeeded or failed. Since the lexicon contains lowercase characters only, the input string is first converted to lowercase.

procedure **Sub_Found**
(VAR LexNode:LexNodeWijzer_Type);
Establishes whether the current character from the input word matches the data-field (char) of the current node *LexNode* in the lexicon trie, and calls Sub_Found recursively with the appropriate pointer field of LexNode^ (YES or NO match) as new parameter. If (current char is last in input word string) and (LexNode^ has field Terminal set to TRUE), then the input word is found: the result for function ZoekLex is set to TRUE, and the word data fields are copied from LexNode^ to InWord_Rec^.

4.5.5 user commands

procedure **Options**

The PROS interactive user can enter the command mode, by typing an asterisk (*) as first character of the input line, followed by a command character. The asterisk triggers routine Options, which executes the user's command. The following commands are implemented:

M	: toggle Monitor watchpoints	(call routine SetMonitor)
L	: enable writing of word Labels to file	(redefine global boolean variable ExtendedOutFormat)
NL	: disable writing of word Labels to file	(redefine global boolean variable ExtendedOutFormat)
I	: new Input file	(close input file; call routine Inq_Open_File)
O	: new Output file	(close output file; call routine Inq_Open_File)
S	: change Screen output	(redefine global variable ScreenMode)
D	: Display program settings	(display enabled watchpoints and values of global variables)
C	: display processing Counters	(call routine Write_Statistics)
P	: spawn new Process	
H	: Help on command options	(display command menu)

4.5.6 monitor

procedure **SetMonitor** ;

Enables or disables monitor watchpoints for each analysis module, according to the user's response. The responded boolean values are stored in a global array. If a watchpoint is enabled for a given module, then the application of each analysis rule within that module is logged in a "monitor file" (and optionally, its non-application as well), and the sentence contents are written to this file, both before and after the module has been applied. If no monitor file exists yet, it is opened by calling routine Inq_Open_File.

procedure **Application**

(naam:LongString_Type; toegepast:BOOLEAN; boodschap:LongString_Type);

For statistical reasons, an array is created, in which each rule is represented by a record containing its name, frequency of application (sentence matches with rule context specification) and non-application (sentence does not match rule context). This routine adjusts the appropriate counter by calling local routine Applied (the "fail" counter is only adjusted if global variable *ApplAll* has value TRUE). Input parameter *toegepast* specifies whether the rule has been applied. This enables you to determine how often a rule fails or succeeds. If a monitor watchpoint has been enabled for the current module, then message string *boodschap* is logged to the monitor file.

procedure **Write_Statistics**

Writes processing statistics to output file and terminal. Counters are implemented for the following objects: sentences, words, FWs, CWs, prosodic boundaries, constituents (between-boundary word strings), orthographic puntuation marks, accents on CWs, accents on FWs, application frequency per rule, CPU time consumption since last initialization. Ratios between some of these counters are also calculated and output.

4.5.7 input and output

The routines in this module refer to the following module-local constants: *AlphaSet* = [**a**..**z**, **A**..**Z**], *NumSet* = [0..9], *PunctSet* = [,.-:;'"()?!]+<blank>.

procedure **LeesZin**

(VAR tf:TEXT; VAR F,L:Wijzer_Type) ;

Reads an orthographic sentence from input text file *tf* into double linked list of word records, starting with *F* and ending with *L*. This is done by creating new word records (with default field values), and calling routine LeesWoord, until the end of sentence is detected. Subsequently, any words consisting of non-printable characters are deleted, and the resulting number of words is counted.

procedure **LeesWoord**

(VAR tf:TEXT; VAR x:Wijzer_Type) ;

Reads a single word from text file *tf*, and stores this word string in a data field of record *x*^. Called by routine LeesZin. If the end of the sentence is detected, and the control variable *EOZin* (declared in LeesZin) is set to value TRUE, so that LeesZin stops appending word records. An End-of-Sentence is detected if :

(a) a word is followed by a question mark of exclamation mark;

(b) a word is followed by a period, unless (b1) the word is a single uppercase character (initial of a proper name, part of abbreviation), (b2) the word contains only digits[8], or (b3) the word is an abbreviation which is stored in the global array *AfkList* [in which case the word is expanded to its full equivalent].

Question mark, exclamation mark, brackets (parentheses), comma, semicolon, and colon are stored as separate words in the sentence [since these may correspond to prosodic boundaries]. Hyphens and quotes within a word string are considered as part of that string (as in **zo'n**, **FNV-er**); otherwise they are stored as a separate word.

[8] In this case, the period is appended to the word if more digits follow the period; these following digits are read from *tf* and also appended.

procedure **SchrijfZin**
(VAR ReportFile:TEXT);
Writes the word strings of the (word records in the) sentence double linked list to text file *ReportFile* (which can also be the terminal screen).

procedure **ShowZin**
(String:LongString_Type; MarkCurr:BOOLEAN; VAR ToFile:TEXT);
Writes the contents of the word records in the sentence double linked list to the text file *ToFile*. All data fields are printed: Word(=string), Synt, (SubVerb), Pros, Acc. For ambiguous words, both feature sets (of the two alternatives) are output. The sentence is preceded by header message *string*; parameter *MarkCurr* controls whether the current word in the sentence should be highlighted.

procedure **Progress**;
Reports what fraction of the input text file which has been processed. This can be useful if Pros processes large text files or runs in batch mode. The fraction is calculated by means of global variables *WordCount* and *FileSize* (supplied by user).

procedure **DisplayFile**
(fn:LongString_Type);
Opens a temporary text file with RMS name *fn*, writes its contents to the terminal screen [user may request repeated display], and closes the file.

5. EVALUATION

As explained in chapter 1, the PROS algorithm aims at establishing the correct 'abstract sentence prosody', viz. accentuation and phrasing. It is assumed that these aspects (or, more properly, the appropriate phonetic correlates of these aspects) make synthetic speech more natural and intelligible. In order to evaluate whether our algorithm achieves this aim, two methods are possible. First, we can compare the PROS output with the accentuation and pausing as produced by a human speaker. Ideally, there should be no difference between the two; any differences found should not disturb the semantic and pragmatic equivalence between the natural and synthetic versions[9]. Secondly, the PROS output can be evaluated from a perceptual viewpoint, viz. by investigating whether synthetic speech provided with accents and pauses derived by means of the PROS algorithm, is more natural and intelligible than some other type(s) of (prosody in) synthetic speech.

5.1 comparison with natural prosody

In order to evaluate our rules for prosodic analysis, a comparison was made between natural speech, and the accents and pauses produced by our PROS algorithm. The natural speech was produced by a professional speaker, who read aloud several texts, the majority of which were originally written as radio news bulletins (and hence, meant to be read aloud). In total, the material consisted of 10766 words [5273 CWs, 5493 FWs] in 600 sentences, grouped into 43 texts. Recordings were made at the Institute for Perception Research (IPO) in Eindhoven. The recordings were transcribed with respect to accents and prosodic boundaries by the two authors (and occasionally, by a third transcriber). If all transcribers agreed, then a word was considered to be accented, otherwise not. The results in Tables 2 to 5 below show the degree of convergence (agreement) between the human speaker and the PROS algorithm. These results are only of limited value, however, since a considerable amount of variation is allowed with respect to 'abstract prosody'. A difference in phrasing and/or accentuation between man and machine does not imply that the machine (the PROS algorithm) has been wrong. The sentence pair (41) below illustrates this variation: both versions are equally acceptable, and roughly equivalent. For the sake of clarity, we will nevertheless use the term *error* for such discrepancies from the naturally produced prosody.

(41a) een aantal ONDERZOEKERS meent overigens ## dat de VRAAG ##
of passief meeroken SCHADELIJK is ##
al LANG positief kan worden BEANTWOORD

[9] One could discriminate between obligatory and optional accents and pauses; differences should be limited to the optional accents and pauses.

(41b) een AANTAL onderzoekers MEENT overigens ## dat de vraag
of PASSIEF MEEROKEN SCHADELIJK is ##
al lang POSITIEF kan worden beantwoord

Table 2: Comparison between the numbers of (possible) prosodic boundaries, as realised by a human speaker and by the PROS algorithm. In total, 10766 [word boundaries] minus 599 [sentence boundaries] = 10167 [intra-sentence word boundaries] were compared.

		HUMAN SPEAKER		
		realised	not realised	total
PROS	realised	845	497	1342
	not realised	474	8351	8825
	total	1319	8848	10167

The data in Table 2 show that PROS predicts 64% of the 'human' prosodic boundaries correctly, and that the same decision is taken in 90% of all relevant cases (error rate 10%). This result remains about the same if we ignore those prosodic boundaries in the PROS output which are taken from orthographic punctuation in the input (e.g. comma's and parentheses; N=627)[10].

Table 3: Comparison between the numbers of accented words as produced by a human speaker and by the PROS algorithm.

		HUMAN SPEAKER		
		+acc	-acc	total
PROS	+acc	3434	852	4286
	-acc	739	5741	6480
	total	4173	6593	10766

With regard to accentuation, the agreement is 85% if all words are pooled. In this connection, it must be noted that the two error types (in Table 3) are not independent. Since accents usually occur in a rhythmic pattern, 'incorrect' accentuation of one word corresponds to an 'incorrect' non-accent on a neighbouring word, as exemplified in the fragments **een aantal onderzoekers meent** and **al lang positief** in (41) above.

In section 3.3.2.1. above, it was explained that FWs are seldom accented. CWs, on the other hand, allow more accentuation variation. Consequently, the agreement in accentuation (as observed in Tables 4 and 5) is considerably higher for FWs (94%) as compared to CWs (77%).

[10] Assuming that these boundaries were also realised by the human speaker.

Table 4: Comparison between the numbers of accented function words (FWs) as produced by a human speaker and by the PROS algorithm.

		HUMAN SPEAKER		
		+acc	-acc	total
PROS	+acc	90	28	118
	-acc	326	5049	5375
	total	416	5077	5493

Table 5: Comparison between the numbers of accented content words (CWs) as produced by a human speaker and by the PROS algorithm.

		HUMAN SPEAKER		
		+acc	-acc	total
PROS	+acc	3344	824	4168
	-acc	413	692	1105
	total	3757	1516	5273

5.2 perceptual evaluation of PROS output

Van Bezooijen (1989) has evaluated the PROS output from a perceptual perspective. Eight texts (total 24 sentences) and eight isolated sentences were fed into the PROS algorithm [version 01-nov-1988]. The output abstract prosodic markers were converted into phonetic prosody (most notably, pitch accents) in diphone speech. In addition, three control conditions were created:

--*random*: the same number of accents as produced by PROS (N=274) were distributed at random over the CWs in the text [low control];

--*subjects*: beforehand, subjects were asked to indicate the accentuation which they considered to be optimal; a word in the stimulus material was accented if 7 out of 12 subjects agreed [high control 1].

--*natural*: a word was accented if a professional human speaker (viz. the same as mentioned above) had accented that word, as agreed by three out of four transcribers [high control 2].

These four accent versions of each sentence were then presented to listeners. Their task was to rate the accentuation of each sentence on a 10-point adequacy scale. Results are summarised in Table 6 below. From these results, Van Bezooijen (1989) concludes that PROS produces sufficiently adequate accentuation, although the PROS output is still defective in several respects.

Table 6: Mean scores on a 10-point adequacy scale, averaged over 32 sentences and 12 listeners, for four accentuation conditions.

random	PROS	subjects	natural
4.6	6.0	7.7	7.4

5.3 error analysis

The majority of the 'errors' (divergences between naturally produced prosody and the PROS output) can be ascribed to one of the following factors.

labeling: An incorrect syntactic word label (e.g. Noun instead of Verb) may yield an incorrect insertion of prosodic boundaries; inappropriate accentuation may result from either the incorrect word label or the incorrectly demarcated prosodic domain. In the follow-up research project, ample attention is dedicated to a solution for this problem, by means of (a) label generation by means of morphological analysis, and (b) improved syntactic disambiguation, using more advanced parsing techniques.

syntax: As explained before, prosodic domains should respect important syntactic and thematic constituents. A considerable amount of phrasing and accentuation errors are due to a violation of this restriction. The phrasing errors are usually rather serious, e.g. incorrect demarcation of subordinate clauses. Such phrasing errors may also propagate into subsequent accentuation errors.

given/new: Any accentuation algorithm should detect which words convey information which is already 'given' to the listener, and then de-accentuate these words. The means which PROS employs to this end are clearly insufficient, since the scope of the algorithm is limited to a single sentence. This deficiency is responsible for the majority of the accentuation errors. Firstly, incorrect accents are assigned to the 'given' words; secondly, the predicate (verbal constituent) is incorrectly *not* accented. In future versions of PROS, several possible solutions to this problem will be investigated (e.g. using a buffer of previously encountered CWs).

remaining: The remaining phrasing and accentuation errors are due to a variety of causes, the most important of which are: (1) idiomatic expressions [yielding idiomatic phrasing and accentuation], (2) incomplete specifications in the PROS lexicon, (3) pragmatically motivated accentuation, which can only be produced with sufficient non-linguistic knowledge. It is in the nature of things, however, that this latter type of error cannot be solved. Hopefully, no algorithm will ever be able to equal human speakers with regard to their knowledge beyond the horizon of linguistics.

6. AUXILIARY PROGRAMS

6.1 procedure within DS

In order to include PROS in a text-to-speech system (viz. the IPO system, based on diphone concatenation; see Van Rijnsoever 1988), our algorithm was implemented as a Pascal procedure (rather than as an independent program), to be called by the main program DS. As explained in section 4.2, this procedure shares the PROS environment and several modules with the main version of PROS. Some adjustments were required, however, in the modules which inherit declarations from the PROS routine itself. The most important changes from the program (described in chapter four) are discussed below.

Firstly, an environment file contains those definitions which are necessary for compatibility with the IPO routines. The VAX/VMS logical name **DSenv** must be equivalent with this environment file DS. Secondly, module PROS was changed, so that PROS was implemented as the following procedure:

procedure **Pros**
(InString:Str; VAR OutString:Str) ;
The data type *Str* is defined in envirnonment file DS. This procedure converts *InString* (read by DS) to a double-linked list (see section 4.4.1), performs prosodic analysis on this sentence (or executes a user command), and writes the resulting sentence to *OutString* (to be output by DS). If *InString* is empty (length zero), then PROS is initialized: global variables are initialized, the lexicon is read into memory, abbreviations are defined, etc.

Thirdly, routines for reading and writing sentences were changed, since this version of PROS communicates with character strings (rather than with text files). These routines (viz. STRING_TO_LIST and LIST_TO_STRING) are declared in module CommRouts, which replaces module LeesSchrijf. Fourthly, minor adjustments were made in the program-specific modules GlobalRouts, Initialize and Options. Several command procedures are available for compiling, linking and testing these DS-versions of the PROS modules.

6.2 partial prosodic analysis

In some cases, it can be useful to break up the prosodic sentence analysis into several parts, which can be performed independently. An auxiliary version PROSPART has been created, which divides the prosodic analysis into three partial sub-analyses:
[1] word labeling (Label_Words, Adj_FWLabels, Adj_VerbLabels, Adj_Labels);
[2] prosodic phrasing (boundary insertion; Insert_Bounds, Adj_Bounds);
[3] accentuation (Select_Accents, Select_VerbAccs, Adj_Accents).

Each partial analysis works independently from the others; the user can specify which part(s) of the analysis is (are) required[11].

For each sub-analysis, the *input* sentence is passed on by the previous sub-analysis, if it was selected. Otherwise, input is read from a text file (for part [1]) or from a file containing word definitions (for other sub-analyses). Likewise, for each sub-analysis, the *output* sentence is passed on to the next sub-analysis, if it was selected. Otherwise, output is written to the standard PROS output text file (for part [3]) or as word definitions to a special file (for other sub-analyses).

The *word definition files* are a powerful tool, since intermediate results can be written to this file, corrected by hand, and fed as "perfect" input to subsequent sub-analyses. Thus, errors in an "early" stage do not propagate throughout the prosodic analysis; this allows for better evaluation of the rules contained in the three sub-analyses.

These word definition files require a very restricted format, with one word on each line, starting in the first column. Sentences are separated by means of one (1) empty line. Information from the data fields of the word records (which together constitute the sentence double linked list) is stored in this file as follows ["_" denotes a blank, "9" denotes any digit 0..9, the field Ambigu is not written as 'TRUE' or 'FALSE', but as a digit where 0=FALSE, 1=TRUE]:

```
              ( TRUE:  99999_^99_^9_^99_99999_^9_^99_^9_ )
Word_Ambigu_  (                ( VERB:  SubVerb_  )       ) _Pros_Acc_99999_^9_^99
              ( FALSE: Synt_   ( UNDEF: SubUndef_ )       )
```

The array of lexical features contains 100 cells. The first 5 of these are always specified, since they contain the codes for word labels (data fields Synt, SubSynt, Pros, Acc; see sections 4.3.2 and 4.4.1). These first 5 features must *always* be read from (written to) the word definitions file [indicated by "99999" above]. The remaining cells are used for additional lexical features. These features are not all specified in the word definitions file[12]. Instead, the rank numbers of the enabled features (flags) are written to this file, preceded by a caret (^). For ambiguous words, both possible feature arrays (Alt1 and Alt2) are specified, followed by the values of fields Pros, Acc, and the features in array LexFeats. Some annotated examples of word definitions follow below:

```
met     0      PREP  FW   FALSE 50101    ^53 ^55 ^59 ^72 ^87
word ambigu synt  pros   acc   5feats   ---more-features---

bemoeilijkt  0     VERB    PART    CW   FALSE 01000
word        ambigu synt subverb pros   acc    5feats   -no-more-feats-

zijn     1         03100                          60100    ^61
word ambigu Alt1: 5feats -no-more-feats- Alt2: 5feats -more-feats-
```

[11] The program has been designed for separate or "adjacent" partial analyses (e.g. [1], [1+2], [2+3]). Violation of this assumption (with partial analyses [1+3]) may yield unexpected results. An analysis with all three sub-analyses ([1+2+3]) is equivalent to the original PROS program, which performs this task more efficiently.

[12] This would result in a zero for each disabled feature.

Upon initialization, PROSPART opens the usual input and output files, and asks which sub-analyses are to be selected. If appropriate, the program asks for an input word definitions file [if first part *not* selected] and/or output word definitions file [if last part *not* selected].

This program shares the environment and many global routines with the original PROS program, as explained in section 4.2. The most important changes in the program-specific modules were the following:

PROSPART: routines ReceiveList and SendList added; these determine whether the sentence must be read from (written to) the standard input (output) text file [by shared routines LeesZin (SchrijfZin), cf. section 4.5.7], or from (to) the word definitions input (output) file;

GlobalRouts: routines GetZin and PutZin added; GetZin reconstructs a double-linked list (sentence) from word definitions read from file; PutZin writes a sentence to a word definitions file;

Options: module deleted; this program does not support user commands.

6.3 context matching during prosodic analysis

The program PROSCON performs a standard prosodic analysis. In addition, however, the intermediate sentence representation can be scanned for a user-supplied target context, after each prosodic analysis module. Let us assume, for example, that we want to know which word(s) precede an Int boundary followed by an inflected Verb. This target context can be roughly described as follows:

	word1	*word2*	*word3*
Word:	?	IntBound	*
Synt:	?	*	VERB:INFL
Pros:	?	PC	FW,CW
Acc:	?	-acc	*
Amb:	?	FALSE	FALSE

After each analysis module, the program verifies whether any fragment of the sentence double linked list corresponds to this context specification. In order to decide this, the context must be matched against each word record in the sentence representation. If a match has been found, then the actual instantiation of the target context (and optionally, the whole sentence) is written to the output file.

In this program, words in the *context* double linked list are declared as follows (note that multiple labels can be specified for each word in the target context):

```
TYPE ContextWijzer_Type   = pointer to ContextWordRec_Type
     ContextWordRec_Type  = RECORD
                          Word:      word string (max 30 chars)
                          Synt:      SET OF syntactic labels
                          Pros:      SET OF prosodic  labels
                          Acc:       boolean: word accented
                          Ambigu:    boolean: ambiguity allowed
                          LexFeats:  array of feature digits
                          Prevw:     pointer to previous word
                          Nextw:     pointer to next     word
```

The matching procedure contains several steps. Firstly, it is checked whether the sentence representation contains sufficient words (from the current word under inspection onwards) to match the target context. This prevents full evaluation of e.g. a three-word context against the last two words of a sentence -- which would never produce a succesful match with the target context. Secondly, for each word in the context specification, the values of the syntactic and prosodic labels[13], and the accent value are checked against the corresponding word in the sentence representation. Finally, the two word strings are matched, using a "wildcard" procedure. If the latter two steps yield a positive result for each word in the target context, then the matching procedure also yields a positive result (viz. boolean value TRUE).

This program has been a very powerful tool in the development of prosodic rules, since it enables the user to find specific word sequences, in the appropriate PROS format, in large text corpora. From the example above, for example, it can be deduced how often a sentence starts with an introductory sub-clause, and how often this sub-clause is properly demarcated by the PROS algorithm.

6.4 divergence between PROS output files

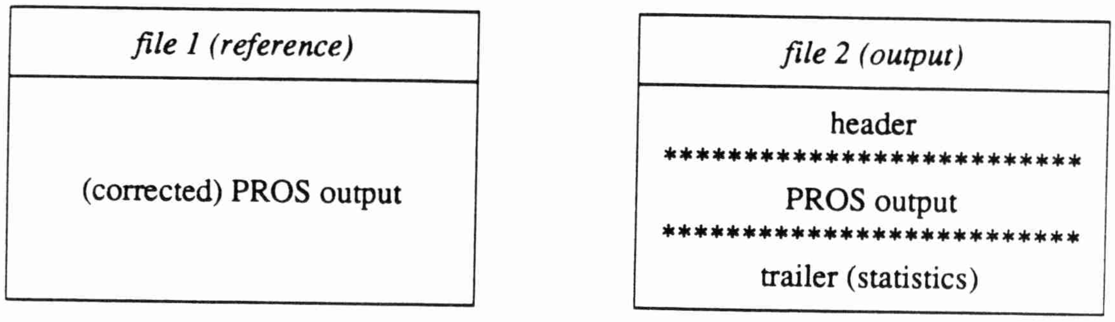
The program DIVERGENTIE compares two output files produced by PROS, and lists the sentences which deviate between the two files to an output text file. This program was used mainly to compare PROS output with a (hand-corrected) "ideal" output version. A divergence is established if (1) the boundary between words is not identical [blank character, Phi boundary, or Int boundary], or if (2) the accentuation of a word differs [as indicated by its typecase].

The first (reference) file may *only* contain PROS output, without any of the standard heading and trailing information normally. The second file, on the other hand, must contain a *header* and a line of asterisks before the PROS output starts, as well as a line of asterisks and a *trailer* (statistics) after the output text. Formats of the two files are schematized below. In both files, the PROS output must be produced with the "short" output option (PROS variable *ExtendedOutFormat* =FALSE), so that only word strings are written, and no syntactic and prosodic labels. Accentuation must be encoded by a word being written entirely in uppercase. Each word must be followed by a blank character, even if it is the last word on a line of text. Output sentences must be separated by an empty line[14].

[13] For the syntactic label, this evaluation yields a positive result if (a) the actual value of the sentence word is contained in the set of possible values in the context word, (b) one of the possible values of the context word is UNDEF [matching all actual labels], or (c) the sentence word is ambiguous, but the feature code of one of the alternatives corresponds to one of the possible syntactic labels in the context word. For the prosodic label, the evaluation proceeds likewise, although with different values and labels.

[14] By default, PROS produces output which matches all these requirements.

Figure 6: Schematized formats of the two PROS output files, as required by program DIVERGENTIE.



Both input files can be provided with a comment string (maximum 80 characters), which clarifies the contents of each file:

file1:	hand-corrected version (21 nov 1989)
file2:	new version with updated accent rules (23 nov 1989)

These comments are written to the output file whenever appropriate. In the output file, differences can optionally be highlighted (by means of escape characters which change the terminal's video settings). In the output file, statistics are given for the number of

(1)-superfluous prosodic boundaries	[not present in file1, but present in file2],
(2)-missing prosodic boundaries	[present in file1, but not present in file2],
(3)-superfluous accents	[words -accent in file1, but +accent in file2],
(4)-missing accents	[words +accent in file1, but -accent in file2].

6.5 command procedures

The logical directory **usr$prosenv** contains some command procedures which facilitate the construction of PROS.

PAS_PROS compiles all source modules for the programme PROS, stores the compiled object modules in a library (**usr$prosenv**:ProsLib.OLB), and finally deletes the object modules.

LINK_PROS links the objects to construct an executable version of PROS.

CH_PROS combines these two procedures: first the editor is called to modify the source code of a module, the resulting updated Pascal module is compiled, stored in the object library ProsLib, procedure LINK_PROS is called (optional), the resulting program PROS is executed (optional), and finally this program is (optionally) stored in an "open" directory, to be used by others.

7. REFERENCES

BAART, J.L.G. (1987) *Focus, Syntax, and Accent Placement: towards a rule system for the derivation of pitch accent patterns in Dutch as spoken by humans and machines.* dissertation Rijksuniversiteit Leiden.

BAART, J.L.G., and J.S. HEEMSKERK (1988) The problem of ambiguity in morphological analysis for a Dutch text-to-speech system. In: *Proceedings SPEECH '88 (7th FASE Symposium), Edinburgh 1988.* 3:959-65.

BEZOOIJEN, R. van (1989) *Evaluation of an algorithm for the automatic assignment of sentence accents in written text.* [Utrecht: Stichting Spraaktechnologie]. SPIN-ASSP Report; 9.

BROECKE, M.P.R. van den, A. AERTS, J. REIZEVOORT, T. VEENHOF, J. LAMMENS, and M. ELSTRODT (1987) Type- and token-frequencies of wordclasses, phonemes and phoneme pairs in Dutch, *Progress Report Institute of Phonetics Utrecht (PRIPU)* 12(1):1-15.

COLLIER, R., and H. 't HART (1975) The role of intonation in speech perception. In: A. Cohen and S.G. Nooteboom (eds.) *Structure and Process in Speech Perception.* Berlin, Heidelberg, New York: Springer. Communication and Cybernetics; 11. 107-21.

COOPER, W.E., and J. PACCIA-COOPER (1980) *Syntax and Speech.* Cambridge MA: Harvard University Press.

COOPER, W.E., and J.M. SORENSEN (1977) Fundamental frequency contours at syntactic boundaries, *J. Acoust. Soc. Am.* 62:683-92.

CUTLER, A. (1982) Prosody and Sentence Perception in English. In: J. Mehler, E.C.T. Walker, and M. Garrett (eds.) *Perspectives on Mental Representation: experimental and theoretical studies of cognitive processes and capacities.* Hillsdale NJ: Lawrence Erlbaum Ass. 201-16.

CUTLER, A., and C. CLIFTON (1984) The use of prosodic information in word recognition. In: H. Bouma and D.G. Bouwhuis (eds.) *Attention and Performance. volume X: Control of Language Processes.* London: Lawrence Erlbaum Ass. 183-96.

FUCHS, A. (1984) 'Deaccenting' and 'default accent'. In: D. Gibbon and H. Richter (eds.) *Intonation, Accent and Rhythm.* Berlin: Walter de Gruyter.

GEE, J.P., and F. GROSJEAN (1983) Performance structures: a psycholinguistic and linguistic appraisal, *Cognitive Psychology* 15:411-58.

GOLDMAN-EISLER, F. (1972) Pauses, clauses, sentences, *Language and Speech* 15:114-21.

GUSSENHOVEN, C. (1984) *On the Grammar and Semantics of Sentence Accents.* Dordrecht, Cinnaminson NJ: Foris. Publications in Language Sciences; 16.

't HART, J., and R. COLLIER (1975) Integrating different levels of intonation analysis, *J. Phonetics* 3:235-55.

HEEMSKERK, J. (1989) Morphological parsing and lexical morphology. In: H. Bennis and A. van Kemenade (eds.) *Linguistics in the Netherlands 1989.* Dordrecht, Providence RI: Foris. AVT Publications; 6. 61-70.

KAGER, R., and H. QUENé (1987) Deriving prosodic sentence structure without exhaustive syntactic analysis. In: J. Laver and M.A. Jack (eds.) *Proceedings European Conference on Speech Technology, Edinburgh 1987.* Edinburgh: CEP Consultants. 1:243-46.

KAGER, R., and H. QUENé (1989) A sentence accentuation algorithm for a Dutch text-to-speech conversion system. In: H. Bennis and A. van Kemenade (eds.) *Linguistics in the Netherlands 1989.* Dordrecht, Providence RI: Foris. AVT Publications; 6. 101-109.

KAGER, R., and E. VISCH (1988) Metrical Constituency and Rhythmic Adjustment, *Phonology* 5(1):21-71.

KLATT, D.H. (1975) Vowel lengthening is syntactically determined in a connected discourse, *J. Phonetics* 3:129-40.

KLATT, D.H. (1976) Linguistic uses of segmental duration in English: acoustic and perceptual evidence, *J. Acoust. Soc. Am.* 59(5):1208-21.

KOOPMANS - van BEINUM, F.J. (1980) *Vowel Contrast Reduction: an acoustic and perceptual study of Dutch vowels in various speech conditions.* dissertation Universiteit van Amsterdam.

KAGER, R., and E. VISCH (1988) Metrical constituency and rhythmic adjustment, *Phonology* 5:21-71.

KNUTH, D. (1973) *The art of computer programming. volume 3: Sorting and searching.* Reading MA: Addison-Wesley.

KRUYT, J.G. (1985) *Accents from speakers to listeners: an experimental study of the production and perception of accent patterns in Dutch.* dissertation Rijksuniversiteit Leiden.

LAMMENS, J. (1989) *From text to speech via the lexicon.* [Utrecht]: Stichting Spraaktechnologie. SPIN-ASSP Report; 7.

LEHISTE, I. (1970) *Suprasegmentals.* Cambridge MA, London: M.I.T. Press.

LINDERT, E. te, C.J. DOEDENS, and H. van LEEUWEN (1989) *Spraakmaker-1.* [Utrecht: Stichting Spraaktechnologie]. SPIN-ASSP Report; 11.

NESPOR, M., and I. VOGEL (1982) Prosodic domains of external sandhi rules. In: H. van der Hulst and N. Smith (eds.) *The Structure of Phonological Representations I.* Dordrecht: Foris. 225-255.

NESPOR, M., and I. VOGEL (1986) *Prosodic Phonology.* Dordrecht: Foris. Studies in Generative Grammar; 28.

NOOTEBOOM, S.G. (1985) A functional view of prosodic timing in speech. In: J.A. Michon (ed.) *Time, mind, and behavior.* Berlin: Springer. 242-52.

NOOTEBOOM, S.G., J.P.L. BROKX, and J.J. de ROOIJ (1978) Contributions of Prosody to Speech Perception. In: W.J.M. Levelt and G.B. Flores d'Arcais (eds.) *Studies in the Perception of Language.* Chichester: John Wiley. 75-107.

NOOTEBOOM, S.G., and J.G. KRUYT (1987) Accents, focus distribution, and the perceived distribution of given and new information: an experiment, *J. Acoust. Soc. Am.* 82:1512-24.

O'SHAUGHNESSY, D.D. (1989) Parsing with a small dictionary for applications such as text to speech, *Computational Linguistics* 15(2):97-106.

RIJNSOEVER, P.A. van (1988) *From text to speech: User manual for Diphone Speech program DS.* [unpublished] IPO Handleiding; 88.

SCHARPFF, P.J., and V.J. van HEUVEN (1988) Effects of pause insertion on the intelligibility of low quality speech. In: W.A. Ainsworth and J.N. Holmes (eds.) *Proceedings 7th FASE Symposium (Speech'88), Edinburgh.* volume 1: 261-68.

SELKIRK, E.O. (1984) *Phonology and Syntax: the relation between sound and structure.* Cambridge MA, London: The M.I.T. Press.

SELKIRK, E.O. (1986) On derived domains in sentence prosody. In: C.J. Ewen and J.M. Anderson (eds.) *Phonology Yearbook.* volume 3. 371-405.

TERKEN, J.M.B. (1985) *Use and function of intonation: some experiments.* dissertation Rijksuniversiteit Leiden.

TERKEN, J., and S.G. NOOTEBOOM (1987) Opposite effects of accentuation and deaccentuation on verification latencies for Given and New information, *Language and Cognitive Processes* 2(3/4):145-63.

VISCH, E.A.M. (1989) *A Metrical Theory of Rhythmic Stress Phenomena.* dissertation Rijksuniversiteit Utrecht.

WINGFIELD, A. (1975) The intonation-syntax interaction: prosodic features in perceptual processing of sentences. In: A. Cohen and S.G. Nooteboom (eds.) *Structure and Process in Speech Perception.* Berlin, Heidelberg, New York: Springer. Communication and Cybernetics; 11. 146-56.

WIJK, C. van, and G. KEMPEN (1985) From Sentence Structure to Intonation Contour. In: B.S. Müller (ed.) *Sprachsynthese: zur Synthese von natürlich gesprochener Sprache aus Texten und Konzepten.* Hildesheim: Georg Olms. 157-82.

8. PUBLICATIONS

R. KAGER and H. QUENé (1987) Deriving prosodic sentence structure without exhaustive syntactic analysis. In: J. Laver and M.A. Jack (eds.) *Proceedings European Conference on Speech Technology*. Edinburgh: CEP Consultants. volume I: 243-46.

R. KAGER (1988) Plaatsing van zinsaccenten en pauzes in spraak. In: M.P.R. van den Broecke (ed.) *Ter Sprake: spraak als betekenisvol geluid in 36 thematische hoofdstukken*. Dordrecht, Providence RI: Foris. 416-27.

H. QUENé (1989) Sprekende Computers, *Toegepaste Taalwetenschap in Artikelen* 33(1):89-94.

R. KAGER and H. QUENé (1989) A sentence accentuation algorithm for a Dutch text-to-speech system. In: H. Bennis and A. van Kemenade (eds.) *Linguistics in the Netherlands 1989*. Dordrecht, Providence RI: Foris. AVT Publications; 6. 101-109.

H. QUENé and R. KAGER (1989) Van tekst naar prosodie, *Informatie* 31(7/8):570-76.

H. QUENé and R. KAGER (1989) Automatic accentuation and prosodic phrasing for Dutch text-to-speech conversion. In: J.P. Tubach and J.J. Mariani (eds.) *Proceedings European Conference on Speech Communication and Technology (EuroSpeech '89)*. Edinburgh: CEP Consultants. volume I: 214-17.

APPENDIX 1 : ANALYSIS RULES IN *PROS*

René Kager and Hugo Quené, 25 october 1989
(c) SPIN/ASSP - RUU/OTS/Fonetiek

NOTATIONAL CONVENTIONS:

{ A,B }	either A or B
(A)L..H	number of occurences of A must range between Low and High
(A)	A is optional; equivalent to (A)0..1
"X"	X is a character string
<X>	X is a label or property
<X>|<Y>	word is ambiguous, label/property is either X or Y
<X>:<Y>	sub-label Y is nested under main label X
*	any character/word, including none
%	any character/word, excluding none
^X	X is a word feature from the lexicon
A[i]...B[i]	conjunctive conditions: A and B must both be true
A[i]...B[j]	disjunctive conditions: either A or B must be true
"A,B"	= "A","B"
<A,B>	= <A>,<B>
<A|B>	= <A>|<B>
<A:B>	= <A>:<B>

In the examples, the relevant part is printed in *italics*; accentuated words are printed in CAPITALS. Where necessary, a plus (+) or minus (-) sign indicates whether a rule has applied at a particular position within the example. Empty positions (e.g. resulting from deletions) are marked with an asterisk (*).

MODULE LABEL_WORDS:

1. LexLabels:

```
*      →      <'spec'>     /     ___
```

condition: if word found in lexicon

'spec' involves data fields Ambigu,LexFeats, and (if Ambigu=TRUE) lexical feature arrays Alt1, Alt2. if word found in lexicon and Ambigu=FALSE, then data fields Synt, (SubVerb, SubUndef), and Pros are specified in accordance with the lexical feature array LexFeats.

2. StrongPV_Format:

```
<Undef> <CW>      →     <Undef|Verb:PV> <CW> /  *     {        "dt"            }
                                                      {<VV> {"tte,dde"} ("n")}}
```

set syntactic label VERB (unambiguous) for all syntactically unmarked content words which (by rule) must be inflected verbs. those words can be detected by the special orthography which results from concatenation of stem and flexion morpheme, yielding an otherwise illegal character combination. words matching this rule are also given the verb-specification <PV> ('PersoonsVorm', i.e., inflected VERB).

--het huis *brand+t*

--zij *hoe+dd+e* zich voor *hoe+d+e+n*

3. SoftPV_Format:

```
<Undef>     <CW>   →     <UNDEF|Verb:PV>   <CW>   /     * { "%t,de,te"              }
                                                          { {"b,t,v,w,z"} "ond"  }
```

add syntactic label VERB for all syntactically unmarked content words which are inflected verbs. inflection can be detected by final morphemes "t,de,te".

--de man *koop+t* een boek

--hij *st+ond* te praten

4. SoftInf_Format:

```
{<Verb>} [1] <CW> →      {<Verb>} [1]  <VERB:Inf> <CW> /      * { <cons> "en" }
{<Undef>}[2]             {<Undef>}[2]                           { %% "aan"     }
```

add syntactic label VERB with sub-specification INF for words ending in morpheme "en". this morpheme can also indicate a plural noun; the requirement for a CONS character before "en" blocks matching of "dien,zien" etc. but leaves e.g. "aaien" unmarked.
--wij *blev+en* drie *dag+en*

5. SoftPart_Format:

```
<X><CW>      «     <X|Verb:Part><CW>
/      ({<Prep>})  {"ge,be,ver,ont"}      { (%)2..8                  }  { "d,en" }
        {<Sat> }                          {((%)2..7 <voc>       )[1]}
         NOT^53                           {((%)2..7 {"p,k,f,s"})[1]}
                                          {((%)2..6 "ch"        )[1]}  {("t")[1]}
condition: label <X> = { <Verb,Undef> }
```

--de koning werd *ver+rad+en*
--de winkel was *uit+ver+koch+t*
--hij vond het wel (-)*be+s+t*

6. Comp_Format:

```
<X>    →    <X|Comp>    /     { <Prep>"dat"   }
                              { "waar"<Prep> }
condition: word must be NOT ambigu, NOT found in lexicon
```

set syntactic label COMP for complementizers, especially for all words that introduce a subordinate clause.
--ik schrijf een boek *na+dat* ik jarig ben

7. Num_Digits:

```
<X>   <CW>   →     <Num> <CW>   /      ("0..9")1..k
```

condition: word must be NOT ambigu, NOT found in lexicon; k = number of characters in word

set syntactic label NUM for all syntactically unmarked content words which consists only of digits
--hij blijft *68* dagen

8. Num_SubString:

```
<X> <CW>     →     <Num> <CW>  /     * ("twee,drie..tien,elf,twaalf,der,veer,
                                        twin,tach,tig,honderd,en,duizend,
                                        miljoen,miljard,half,tal")1..k
```

condition: word must be NOT ambigu, NOT found in lexicon

- set syntactic label NUM for all syntactically unmarked content words which consists exhaustively of orthographic substrings representing numeral morphemes.
--hij blijft *drie* dagen
--de *elf* zit te *zeven* op het j+acht
--de *acht+en+tach+tig+duizend* inwoners

9. Adv_Format:

```
<UNDEF> <CW>      →     <ADV> <CW>
/     NOT {"een, het"}       { * {"lijk(er),ig,isch,baar,end,zaam,dag,middag,
          {<ART,PREP>}             avond,nacht,morgen,ds,loos,daar,hier"}      }
                             { "onge" * {"t,d,en"}                              }
                             { * ("%")4..k {"eel,aal,ief"}                      }
            1                           2
```

conditions: word must be NOT ambigu, NOT found in lexicon; focus is second word in above specification

set syntactic label ADV for adverbia, which can be recognised by special adverbial suffixes. detection of ADVerbia can be useful for phi-construction and accentuation. target word may not be preceded by PREPosition or ARTicle.
--als ik *sportief* rij ...
--ik drink over het algemeen *steeds* koffie

10. Adj_Format:

```
<UNDEF> <CW>          →     <UNDEF:ADJ>
/       {"een,het"}         * { "end,isch,air,ent,ant,lijk,ig" }
        {   <ART>   }
             1                           2
```

conditions: word must be NOT ambigu, NOT found in lexicon; focus is second word in above specification

MODULE ADJ_FWLABELS :

This module dis-ambiguates words which are specified as ambiguous in the lexicon, e.g. "dat,zijn". Disambiguation requires that the adjacent words are disambiguated as far as possible; some elementary disambiguation is therefore performed by the first two routines. Within this module, the rule ordering is very sensitive. In general, routines are ordered according to their context requirements: disambiguation requiring little context information is performed first. Cases for which syntactic label information from surrounding words is necessary, are disambiguated as late as possible; this syntactic information may itself be unreliable. Disambiguation is performed by calling a global routine (SelectVariant), with the intended resulting word properties as input parameters.

11. NonVerb_over_Undef:

```
<UNDEF> |           <X>                  →     <X>
            NOT<{VERB,UNDEF}>
```

if one of the two variants is UNDEF, and the other variant is neither <Verb> nor <Undef>, then select the other variant. This is more or less an artefact of our strategy to store "new" alternatives (esp. ADV,NUM) as syntactic variants, instead of immediately "overwriting" the default UNDEF label. VERB disambiguation is performed by more advanced routines.

12. SubUndef_over_NoSubUndef:

```
    <UNDEF1>        |     <UNDEF2>         →     <UNDEF2>
NOT <{NOUN,ADJ}>          <{NOUN,ADJ}>
```

if both variants are syntactically UNDEF, then select the variant to which additional sub-labels are attached.

13.1. VERB_if_VerbCues:

```
      "zijn"               →     <PRON>       /     { [NIL]   }  ___
<PRON> | <VERB:INF>                                 { <PREP> }
```

sentence-initial "zijn" cannot be VERB, must be PRON. this rule hypergeneralises for Yes/No-questions. after a genuine PREP (no SATellite), "zijn" is assumed to be a modifier PRON within the PP: [PREP "zijn" NP]PP.
--hij wil de beste van +*zijn* klas -*zijn*

13.2. VERB_if_VerbCues:

```
      "zijn"               →     <VERB:INF>  /    ___   { [NIL]          }
<PRON> | <VERB:INF>                                     { <VERB:PART>    }
                                                        { <PC>, <FW>     }
                                                        { <ADV>, <NUM>   }
```

one of the specified righthand contexts is sufficient to make "zijn" a VERB. ADV,NUM must be unambiguously known (cf. rule NonVerb-over-Undef).

--hij wil de beste *zijn*	[NIL]
--hij wil de beste *zijn,* maar haalt het niet	<PC>
--hij wil de beste *zijn maar* haalt het niet	<FW>
--hij wil de beste *zijn geweest* maar haalt het niet	<Verb:PART>
--hij wil de beste van -*zijn* klas +*zijn*	

13.3. VERB_if_VerbCues:

```
      "zijn"               →     <PRON>       /     ___
<PRON> | <VERB:INF>
```

default (context-free) disambiguation if previous routines have failed to work.

14.1. Num_before_Partitive:

```
"een" <ART|NUM>    →    <NUM> /    ___  { ^52   }
                                        { [NIL] }
```

disambiguate "een" as NUMeral, if it is followed by a partitive construction, referring to quantities. indicators for partitive constituents (e.g. "der") are marked with feature ^52 in the lexicon.

14.2. Num_before_Partitive:

```
"een" <ART|NUM>         →    <ART> /    ___
```

default disambiguation if previous rules have failed to work.

15.1. Pron_before_FW:

```
"het" <ART|PRON>   →    <PRON>     /    ___  { [NIL] }
                                             { <FW>  }
```

"het" cannot be used as ARTicle in sentence-final position. the structure of NP constituents (with "het"=ART) requires the next word to be NOUN,ADJ, or ADV. such words are seldom FW, so the presence of a following FW indicates that "het" must be PRONoun rather than ARTicle. this rule hypergeneralises for eg. "HET ZEER zwaar getroffen land"

15.2. Pron_before_FW:

```
"het" <ART|PRON>   →    <ART> /    ___
```

default disambiguation if previous rule has failed to work.

16.1. Prep_before_ART:

```
"gezien" <PREP>|<Verb:PART>   →    <PREP>     /    ___  { <ART> }
                                                        { ^61   }
```

disambiguate "gezien" as PREP or VERB:PART. this rule requires that the syntactic class of following "een,het,zijn" is known, so the rule must be ordered after Num_Before_Partitive ("een"), Pron_Before_FW ("het"), and Verb_if_VerbCues ("zijn"). Feature ^61 indicates possessive pronouns.

16.2. Prep_before_ART:

```
"gezien" <PREP>|<Verb:PART>   →    <VERB:PART> /    ___
```

default disambiguation if previous rule has failed to work.

17. Pron_before_FWVerb:

```
"dat" <COMP|PRON> →    <PRON>      /     ___    { <VERB,FW> }
```

--maar *+dat* *is* wijsheid achteraf

18. Pron_after_PronCue:

```
"dat" <COMP|PRON> →    <PRON>      /     { <VERB,FW> }     ___        { [NIL]           }
                                         { "zijn"    }                { NOT{<ART,PRON>} }
                                         { ^66       }                { "het,er",^21}   }
```

--maar nu *is* *+dat* wijsheid achteraf
--dat *hij* *+dat* liever doet
--het kan ook *blijken* *-dat* *de* mensen niet komen"
--hij belooft *mij* *-dat* niemand zal komen"

19. Pron_before_PC:

```
{"het,dit,dat"} <PRON|X> → <PRON>   /     ___    { [NIL] }
                                                 { <PC>  }
```

words cannot be used as ARTicle ("het") or COMPlementizer ("dat") in these contexts.

20. Pron_between_PREPCONJ_CW:

```
{"dat"}[1] <COMP|PRON>  →   <PRON>   /   {<PREP,CONJ>}   ___   (<CW>)[2]
{"die"}                                  {  ("om")[1] }

        - {PREP,CONJ} + "dat" + CW
          "om"        + "dat" + *
          {PREP,CONJ} + "die" + CW
```

words "die,dat" must be pronominal instead of Complementizer, if they occur between PREP-or-CONJ and CW. This is based on a count of all occurrences of "die,dat", showing that the majority of the <PRON> interpretation occurs when preceded by a preposition. (low-freq) hypergeneralization as in example (3) is taken for granted.
--*in dat* geval ...
--*en die* man zei toen ...
--ik loop de winkel *in die* schoenen verkoopt   (wrong)

21. Pron_after_COMP:

```
{"dat,die"} <COMP|PRON> →   <PRON>   /   { <COMP>   }   ___
                                         { <COMP|X> }
```

22. Comp_after_CompCue:

```
"dat" <COMP|PRON> →   <COMP>   /   { <ADV,PREP,PRON> }   ___
```

23. Undef_after_COMP:

```
NOT{"dat,die"} <COMP|UNDEF>  →   <UNDEF>   /   { <COMP>   }   ___
                                               { <COMP|X> }
```

ambiguous "dat,die" may not be set to UNDEF, but to PRON. for these two words, special contexts apply. therefore, they are excluded here and treated separately in routine Pron_after_COMP.

24. Comp_between_VERB_FW:

```
"of" <COMP|X>     →     <COMP>     /     <VERB>     ___     { <FW>     }
                                                              { <FW|FW> }
```

25. Default_CONJ:

```
"of" <CONJ|X>     →     <CONJ>     /     ___
```

default disambiguation if the previous rule has not been applied (X=COMP). this rule must be ordered before Default_COMP, because "of" must be CONJunctive by default.

26.1. Conj_after_COMMA:

```
"maar" <CONJ|ADV> →     <CONJ>     /     ","     ___
```

26.2. Conj_after_COMMA:

```
"maar" <CONJ|ADV> →     <ADV> /          ___
```

ambiguous "maar" is only used as CONJ (with preceding IntGrens) if it it preceded by a comma ",". otherwise, used as ADV (modifier), and hence no preceding IntGrens inserted.

27.1. Comp_before_PRONART:

```
"wat" <COMP|PRON> →     <COMP>     /     ___     { <ART,PRON> }
                                                  { "het"      }
```

27.2. Comp_before_PRONART:

```
"wat" <COMP|PRON> → <PRON>     /     ___
```

this rule must be ordered before Default_COMP, because "wat" must be PRONoun by default.

28. Default_COMP:

```
<COMP|X>     →     <COMP>      /      ___
```

default disambiguation if the previous rules have not been applied. this rule prevents ambiguities involving COMP to be output. this rule must be ordered after Default_Conj ("of") and Comp_before_PronArt ("wat").

MODULE ADJ-VERBLABELS:

this module processes words for which a second label VERB has been specified in Label_Words, resulting in ambiguous words. because label rules over- and under-generalize, and because lexical ambiguities can be expected, the following rules use contextual information to disambiguate the syntactic word labeling of verbs. finally, remaining ambiguities are resolved by assuming the VERB label to be correct by default. Rules 31..42 are called by a local routine (NegFilter). This routine disambiguates a word as UNDEF if one of the filtering routines has applied.

29. Verb_After_ER_Adv:

```
<X|VERB>     →     <VERB>      /      "er"*
                                      <ADV,FW>     ___
```

this rule must be ordered PRIOR to NegFilter\TwoCWs. it is a separate routine, since this rule should work before the rules called by NegFilter rules: it is a "positive" rule (resolving ambiguities as VERB).
--hij wil dat ik de groente *erin gooi*
--ze hadden *ertoe besloten* weg te gaan

30.1. InfFilter:

```
<CW> <X|<VERB:INF>>     →     <VERB:INF>   /      "te"    ___
```

disambiguate words with INF sublabel as VERB:INF, if preceded by "te"

30.2. InfFilter:

```
NOT "*en"            →    <UNDEF>     /     "te"  ___
<X>|<VERB>
```

VERBs with sublabels <PV> or <PART> are incorrectly labeled after "te"; assume that VERB label is wrong and disambiguate as UNDEF (after "te" follows either VERB:INF or UNDEF:ADJ).

31. Caps:

```
<X|VERB>    →    <X>    /    <capital> *
```

words which begin with a capital can not be verbs, but must be a proper name. no exception for sentence-initial words, because these are rarely VERBs (only in Yes/No-questions).

32. Samenstelling:

```
<X|VERB>    →    <X>    /    "*-*"
```

a word containing a hyphen is a compound, which is seldom VERB.

33. TooLong:

```
<X|VERB>    →    <X>    /    "%"(14) "*"
```

words longer than 13 characters are probably compound NOUNs; VERBs are usually shorter. this filter produces errors in VERBs like "herprogrammeren", but filters many wrong VERB labels in eg. "wapenhandelaren" etc.

34. NounCues:

```
<X|VERB>    →    <X>    /    (NOT ^81)[1]    { <Art,Conj,Comp> }    ___
("*en")[2]                                   { <PREP>[1]       }
                                             { <NUM>[2]        }
                                             { ^13             }
```

some ambiguous words must be plural nouns, because the preceding word can only (or mainly) precede nouns in Dutch. lexicon feature ^81 indicates [+R] words (eg. "er,daar,hier"); ^13 indicates NUMeral nouns "geen, aantal", etc.

```
--wij bleven drie +dagen
--hij heeft het ergens uit -gehaald
--hij bleef er in -hangen          ^81 + PREP + <X|VERB>
```

35. Flexions:

```
<X|VERB>    →    <X>    /    [NIL][1]    "*%<cons>e"        ___
                                         NOT "*%je"
                                         { NOT <cap>*  }
                                         { "*s(ch)e"   }
                                         { "*"[1]      }
```

if the preceding word ends in "e", that word is (probably) an inflected adjectivum, and the target word is the plural noun it is related to -- instead of VERB. all two-letter words ending in "e" in Dutch are (FW) article, preposition or pronomen personale, which CAN precede VERBs. therefore, it is required that the preceding word contains three or more characters. "de" and "te" block the VERB label in other rules.

--nu komen de *vette +jaren*
--ben *je -belazerd*
--door de gemeente *Ede -uitgekozen*
--de *Olympische +gedachte*

36. PossPron:

```
<X|VERB>    →    <X>    /    <PRON> ^61  ___
```

words preceded by an unambiguous pronomen possessivum can not be VERB, but must be plural nouns. Lexical feature ^61 marks possessive PRONouns.

--ze knipt *mijn haren* maandelijks

37. TwoPossVerbCWs:

<X|VERB> <CW> → <X> / ___ {<X|VERB>,<VERB>} <CW>

in a sequence of two CWs both syntactically ambiguous (with one of the alternatives being VERB), the first one has been mislabeled and should be plural noun <UnDef>. this rule is statistically motivated, not based on any linguistic fact.

38. PrevArtPart:

<X|VERB> → <X> / <ART> <VERB:PART> ___

if the <Verb:PART> is preceded by an ARTicle, then this PARTiciple is used as an adjective. the target word is a plural NOUN, head of an NP.
--de *verboden vruchten* smaken het lekkerst

39. PlurNoun_INGEN:

<X|VERB> → <X> / "*<V>*ingen"

the target word must be a plural noun if it contains (a) a PLURAL suffix "en", preceded by (b) a nominalisation <NOUN> suffix "ing". the preceding vowel <V> is required, since a stem syllable must precede the offset.
--dat in Frankrijk +*verkiezingen* worden uitgeschreven
--hij kan het niet laten +*aanwijzingen* te geven
--hij wil ons -*dwingen*

40. PrevArtAdj:

<X|VERB> → <X> / <ART> * { "end,isch,air,ent,ant,lijk,ig" } ___

if the preceding word is an adjective (which can be deduced from its adjective suffix) and preceded by an ARTicle, then the target word is a noun, head of an NP, instead of a VERB.
--een smoezel*ig vest*
--een sociaal-democrat*isch nest*

41. PrevAdv:

<X|VERB> → <X> / ^82 ___

if the target word is preceded by an intensifying adverb (lexical feature ^82) then the target word is (probably) an adjective or noun, instead of a VERB.
--in een *heel hecht* verband
--dat ik me aan zoiets *minder hecht* (wrong)

42. TwoCWs:

<X|VERB> → <X> / NOT {comma,^62} ___ <CW>
<Y|CW>

statistically speaking, CW/VERBs seldomly occur before CWs. there are two exceptions: after a comma, or after a nominative=subject pronoun (lexical feature ^62).

43. Verb_after_PronSubj:

<X|Y> → <VERB:PV> / [NIL] (<PC>) ^62 ___

a special problem occurs in detecting uninflected (stem) VERBs, which can not be detected on formal properties. however, such VERBs MUST occur in the second word position in the sentence, if the first is a one-word pronominal subject. if so, then the following word must be VERB, regardless of its properties. only the PRO-Noun "het" may be be excluded from the lefthand context. in order to accomodate direct speech, quotes <PC> may precede this first pronomen.
--*'ik +snij* mezelf in de vingers', sprak de kok

44. PV_over_PART:

<VERB:PV> | <VERB:PART> → <Verb:PV> / {"be*,ge*,ver*,ont*"}"*t"

the <PART> sublabel may have been added incorrectly (in module Label_Words) for words which are morphologically ambiguous <PV|PART>. the <PV> label is preferred in those cases. this preference must be made explicit, since by default the last-one-added of the two variants is selected, if variants are identical (both VERB in this case). this last-one-added variant would otherwise be the (less frequent) <PART> variant.

45. Cash_VERBs:

```
<X|VERB>      →      <VERB>      /      ___
```

default disambiguation for ambiguous VERBs, after previous rules have been applied.

MODULE ADJ_LABELS:

adjustement of the syntactic labeling of words which can only be disambiguated AFTER previous disambiguation routines. the previous routines provide the necessary context for this module. in addition, existing labels are modified.

46.1. Undef_after_VERB:

```
"dan" <ADV|UNDEF> →     <UNDEF>      /      <VERB>      ___
```

UNDEF = voegwoord van vergelijking.

46.2. Undef_after_VERB:

```
"dan" <ADV|UNDEF> →     <ADV> /      ___
```

default disambiguation if the previous routine has not been applied.

47. Nouns_UNDEF:

```
<UNDEF>      →      <UNDEF:NOUN>      /      { ("%")14 "*"                    }
<CW>                                         { "*" {"heid,ie(s),je(s),
                                                    sel(s),ing(en)"}    }
```

add a sub-label NOUN to UNDEF words, if they have the proper format for NOUNs.

48. Plural_UNDEF:

```
<UNDEF>      →      <UNDEF:NOUN>      /      "*%%"
                    ^99                      { "*ingen,*jes,*sels" }
                                             { "*en" NOT<NUM>      }
                                             { ( NOT"<cap>*")[1]
                                               (     "*<C>s" )[1]  }
```

add a sublabel NOUN and add lexical feature ^99 (indicating plural) to UNDEF words, if they have the proper format for plural NOUNs. <C>=<Consonant>.

49. Adj_UNDEF:

```
"*e" <UNDEF>        →      <UNDEF:ADJ> /          ___          NOT<FW>
NOT{"*ie,*ee,
*oe,*je,te"}
```

following word must belong to same NP as ADJective, hence following word cannot be FW.

MODULE INSERT_BOUNDS:

this module inserts prosodic boundaries (ZinsGrens=#### , IntGrens=### , PhiGrens=##) between words in the sentence.

50. Zins_Around_Sentence:

```
0      →      ZinsGrens <PC>      /      { [NIL] ___ * }
                                         { * ___ [NIL] }
```

insert utterance boundaries before the first word and after the last word of the sentence.

51. V2Infl:

```
V2Infl      →     TRUE  /     (NOT"te")[1]              ___
                                                      <Verb>
                                                    { <PV>,<Inf>[1] }
```

determine whether focus word is Verb in second position (i.e., inflected verb, main VP).

52. Int_After_VerbCluster:

```
0     →     IntGrens <PC>     /     <VERB>         ___     NOT <Verb>
                                    NOT V2Infl             {<CW>,<FW>}
```

insert an intonational boundary between a verbal cluster which terminates a subordinate clause (hence: exclude inflected VERBs, called V2Infl's), and a following non-VERB.
--ik vroeg hem te *komen* ### *met* al zijn vrienden
--zij heeft erin *toegestemd* ### *een* plan te schrijven
--ze vertelde me te *hopen* ### *morgen* terug te komen

53. Int_Replace_PC:

```
* <PC>      →     IntGrens <PC>     /     { "punctuation marks" }
```

change punctuation marks "(-;:,)" to intonational boundaries. single- or double-quotes (ASCII 34,39) are excluded from this replacement!

54. Int_Before_Comp:

```
{ 0 }                        →     IntGrens <PC>
{<PC> NOT-ZinsGrens }
/     NOT {<COMP,CONJ>}            ___         <COMP>
                                               NOT "dat"
      (NOT<PREP>)[1]                           (NOT^31)[1]
```

separate nested/subordinate clauses (indicated by COMPs) from context. consecutive COMPs or CONJunctiva are grouped in the righthand constituent, by blocking insertion between them. the conjunctive contexts prevent S-

bars to be separated from the PREP with which they form a PP (Prep + Sbar). exclude the COMP "dat"; these cases are treated separately (see rule Int_before_DAT below).
--ik bezoek mijn broer ### *die* in Amsterdam woont
--ik vraag ### *of* (-) *wie* de schoen past hem aantrekke
--ik vraag ### *welke* deur ik moet openen
--ik vraag *door* (-) *welke* deur ik moet lopen"

55. Int_Before_DAT:

```
0    →    IntGrens <PC>    /    (<VERB:PV><CW>)[1]    ___    "dat"
                                                          (<COMP>)[2]
```

"dat" may have 'missed' its correct COMP label, because the context may still have been ambiguous at the time of disambiguation. therefore, one context in which "dat" is most often COMP is still explicitly checked in this routine, as condition [1].
--de man *ontkende* ### *dat* hij gek was
--de man ontkende *uitvoerig* ### *dat* hij gek was

56. Int_Before_CONJ_Const:

```
0    →    IntGrens <PC>    /    NOT<PC>    ___    <CONJ>    {<FW>,<VERB>}
```

this rule insert intonational boundaries before a conjunctive ("en,maar,want..") which introduces a juxtaposed sentence. this context with CONJ linking two juxtaposed sentences can be selected by requiring either an FW or a VERB following the CONJ -- as some statistics have revealed.
--hij wil brood ### *en*<CONJ> *loopt*<VERB> dus naar de bakker
--hij staat op ### *en*<CONJ> *bij*<FW> de bakker koopt hij een brood

57. Int_Between_Verbs:

```
0    →    IntGrens <PC>    /    <VERB>            ___    <VERB>
                                (NOT<Part>)[1]           (NOT<FW>)[2]
                                <CW>[3]                  <PV>[4]
```

```
NO insertion:           <VERB:PART> ___   <VERB,FW>
                        <VERB,FW>   ___   <VERB:PV>
```

insert an IntGrens between subordinate clause (ending in VERB) and main clause (starting with VERB), in order to demarcate the two clauses.

--de voorraden die ons *resten* ### *blijken* onvoldoende

--wat mij *gebeurd* (-) *is* ### *benauwde* me vreselijk

58. Int_Before_ExtraConst:

```
0    →    IntGrens <PC>
/    (<Verb>)1..n     {<Verb>,<PREP>}   ___   { <Conj,Prep> }
                                                NOT^53
```

this rule inserts intonational boundaries before an extraposed constituent, i.e. a constituent following the main clause of the sentence. application of this rule is excluded in the following context:

ZinsGrens + X + <Verb> ___ Y

where X represents a single word : if the main clause only consists of a one-word subject and one verb, then there is no need for strong prosodic demarcation from the second part of the sentence (Gee & Grosjean 1983). at least one verb must precede the rule focus (either directly = in lefthand context, or indirectly), i.e., the focus must follow a nuclear sentence.

--de man *koopt* ### *en* verhuurt goederen

59. Int_Before_INFConst:

```
0    →    IntGrens <PC>    /    NOT {ZinsGrens}  ___   <PREP>     {  "te"  }
                                    {IntGrens }          ^54      { <PREP> }
```

this rule inserts intonational boundaries before a "beknopte bijzin"; prepositions which can start such a construction are marked with feature ^54 in the lexicon.

--hij steekt over ### *zonder uit* te kijken

--hij steekt over ### *om te* kijken

60. Int_Before_DAN:

```
0    →    IntGrens <PC>    /    ___    "dan"
                                     <UNDEF>
```

the word "dan" is lexically ambiguous, as either ADV or UNDEF ("voegwoord van vergelijking", comparative pronoun). the latter case results in non-application of Int_Before_ExtraConst, which is not correct. therefore, this rule inserts an IntGrens before this latter type of "dan" <UNDEF>. "dan" is set to <UNDEF> by module Adj_FWLabels iff preceded by <VERB>.
--dat de auto sneller om de hoek rijdt ### *dan* rechtuit

61. Phi_Before_FW:

```
0    →    PhiGrens <PC>    /    <CW>    ___    <FW>
                                           NOT<Verb>
```

this rule accounts for most of the boundary insertion, and follows directly from prosodic domain theory. head of any prosodic domain is the nuclear constituent content word, and function words on the lefthand side (for Dutch) are included in the domain. in other words, a phi boundary is to be inserted between a content word and a following function word, since the latter belongs (with its following content word) to a different prosodic domain.
--de *auto* ## *met* drie *deuren* ## *zonder* ramen ...

62. Phi_Around_VerbString:

```
0    →    PhiGrens <PC>    /    +α<Verb>    ___    -α<Verb>
                                NOT <PC>           NOT <PC>
```

for several reasons (see text), VERBs are to be grouped in one domain, closed off by phi boundaries.
--ik ## *wil gaan schaatsen* ## op het ijs
--de man ## *koopt* ## een boek

63. Phi_Before_Adv:

```
0    →    PhiGrens <PC>    /    <CW>  ___   <ADV>
```

sometimes the preceding rules generate a long string of CWs, all belonging to one prosodic domain. this rule (and others) provides some means to split one long prosodic domain into smaller ones by inserting phi boundaries at major breakpoints, viz. before ADVerbs (usually NOT complements to the LEFThand constituent).
--ik drink ## over het algemeen ## *steeds* koffie

64. Phi_After_Plural:

```
0     →     PhiGrens <PC>     /     <UNDEF:NOUN>     ___     NOT<VERB>
                                    ("%")4 "*"               <CW>
                                    <CW>
                                    ^99
```

this rule also provide means to split a long prosodic domain into two phi domains. it is assumed that plural nouns establish the head of a prosodic domain (which includes preceding FWs), and a boundary can be inserted after this head. plurals are detected by module Adj_Labels; indicated by feature ^99.
--dat de dak*en*(+) ## kapot waren
--dat de Amerikan*en*(+) ## honger hebben
--dat het Nederland*s*(-^99) moeilijk is

65. Phi_After_Name:

```
0     →     PhiGrens <PC>
/     NOT{ZinsGrens}      "<cap>*"        ___     NOT{"<cap>*"}
         {IntGrens }      NOT"*s(e)"                 { <PC>   }
```

analogous to Phi_After_Plural, this rule inserts a phi boundary after a string of words which begin with capitals. the name is considered to be an NP head. again nationality adjectiva are excluded.
--dat *Cicero* ## *zelfmoord* pleegde
--dat *Hollandse* (-) *kaas* lekker is

66. Phi_After_Noun:

```
0     →     PhiGrens <PC>     /     <UNDEF:NOUN>     ___
```

analogous to Phi_After_Plural, this rule inserts a phi boundary after a noun (indicated by nominal suffixes), functioning as NP head. NOUNs are detected by module Adj_Labels

MODULE ADJ_BOUNDS:

this module adjusts the strength and location of prosodic boundaries. insertion rules occasionally generate boundaries at incorrect places, sometimes as a result of rule interaction. the situation is corrected in this module.

67. NoPhi_After_FW:

```
PhiGrens    →    0    /    <FW>         ___
                           NOT<VERB>
```

(nonverbal) FWs can never establish the head of a prosodic domain. therefore, they have to cliticize to the righthand domain, by means of boundary deletion.
--*ik* * was verbaasd ### over ons succes

68. NoPhi_Before_FWInt:

```
PhiGrens    →    0    /    ___    <FW>   {ZinsGrens,IntGrens}
```

if an FW establishes a prosodic domain on its own, it should cliticize to its lefthand adjacent domain, by means of boundary deletion.
--dat de obers ## boos * *zijn* ### om niets

69. NoInt_Within_Idioms:

```
IntGrens    →    0    /    (1)  "om"   ___  "te"
                           (2)  "aan"       "toe"
                           (3)  "af"   ___  "en"      ___  "toe"
                           (4)  "tot"       "en"           "met"
                           (5)  "nu"        "en"           "dan"
                           (6)  "van"       "binnen"       "uit"
                           (7)  "van"       "buiten"       "uit"
```

delete prosodic boundaries within idiomatic expressions which belong to a single domain.

70. NoInt_Before_IsoPrep:

```
IntGrens    →    PhiGrens    /    ___   {<PREP,SAT>}    {ZinsGrens,IntGrens}
```

an Int-boundary may be inserted between a VERB and its following particle (e.g. by Int_Before_ExtraConst). this rule changes this boundary to a Phi-boundary; this enables for future accentuation of the particle. this rule must be ordered BEFORE NoInt_Before_FinalFWs, as otherwise the latter rule would delete the boundary preceding the verbal particle (thus blocking future accentuation).

71. NoInt_Before_FinalFWs:

```
IntGrens    →    PhiGrens    /    ___   (<FW>)1..n  ZinsGrens
```

a sequence of sentence-final FWs is not a separate prosodic I-domain, but must be cliticised to its preceding domain. at least one FW must occur in the righthand target context.

72. NoInt_After_V2:

```
IntGrens    →    PhiGrens    /    IntGrens    (<VERB>)1..n    ___    <PREP>
```

this rule corrects a hypergeneralisation of rule Int_Before_ExtraConst, which inserts an IntGrens between VERB and PREP. this boundary is incorrect if the PREP introduces an adverbial PP, while a subordinate sentence is used as subject. the subject sentence is indicated by IntGrens preceding the string of VERBs; at least one VERB must follow the IntGrens in the lefthand target context.

--de bibliotheek ### die 20 miljoen kostte ### *wordt gesloten* ## *wegens* stakingen

73. NoInt_After_Prep:

```
IntGrens    →    0    /    <PREP>        ___
                           ^53
```

the PREPositions with feature ^53 (in the lefthand context) can never be verbal particles. consequently, these words should constitute a single domain with the following NP, by means of deleting the erroneous boundary.
--ik hou niet *van* * bietjes
--ik kan niet *zonder* * zei de man (wrong)

74. NoInt_After_IsoPrep:

```
IntGrens    →    0    /    IntGrens   {<PREP,SAT>}    ___
```

let the 'isoprep' (particle,satellite) cliticise to the righthand domain, i.c. verbal cluster.
--hij belooft ### *mee* * te komen

75. Clit_VerbPron:

```
   1           2         →    2    1    /    <VERB>     ___
PhiGrens    ^{63,88}
```

an object or subject personal pronomen (indicated by feature ^63 for pers.pron., feature ^88 for "er") belongs prosodically to the VERB domain. therefore, the positions of the phi boundary and this pronomen are interchanged.
--ik was * *hem* ## te vlug af"

76. ComplexSubjNP:

```
<PC>  →    IntGrens <PC>    /    X             Y          Z    ___    <VERB>
                                 NOT{<VERB,COMP>}
```

the unspecified context X+Y+Z must contain either (a) 3 or more CWs, or (b) 2 CWs and 2 or more PhiBounds (2 CWs, each constituting a separate Phi domain), in order for this rule to apply. preceding VERB or COMP indi-

cates a subordinate clause, instead of subject NP. the effect of this rule is that a single subject NP constituting more than 1 Phi domain is raised to the Int level, by changing the boundary to IntGrens. internal PhiBounds in the subject NP are not deleted.
--de *roodbruin getinte zomerjurken ### zijn* uiterst populair
--de *woordvoerder* ## van het *ministerie ### deelde* mee ...

77. Purge_DoubleBounds:

```
<PC>   →   0     /     (<PC>)        ___    (<PC>)
```

delete the weakest of two adjacent boundaries. if both boundaries have the same strength (viz. length in chars) then the second one is deleted. this rule corrects anomalies created by incorrect rule interactions.
in some PROS programmes, periods and question marks (sentence terminators) are included as words in the sentence linked list. these sentence terminators are not deleted: <PC> NOT IN [".?"].

MODULE SELECT_ACCENTS:

78. Set_LexAccs:

if the target word has previously been found in the lexicon, then the word features ^1..^5 have been copied from the lexicon to the word data field LexFeats[1..5]. feature ^4 = LexFeats[4] indicates the accentuation status of this target word. this routine translates the feature value '1' to the value 'TRUE' for the data field ACCentuation.

79. Acc_IsoPrep:

```
<-acc>        →  <+acc>  /     {<ADV,PRON,PREP>} (PhiGrens)  ___   {IntGrens }
{<PREP,SAT>}                   {    <VERB,NUM>   }                 {ZinsGrens}
NOT{"te,van"}                  NOT^81
```

this rule accentuates stranded particles (PREPositions) at righthand domain edge. the particle is NOT accentuated if the preceding word has feature ^81 ("er,daar,hier," etc).

```
--ik neem er drie +MEE ####
--ik zit hier -mee ####
```

80. Acc_CWs:

```
<-acc>      →  <+acc>  /      <CW>, NOT<VERB>
```

default accentuation of non-VERBal CWs. accents are removed by subsequent rules in module Adj_Accents.

81. Acc_Num:

```
<-acc>      →  <+acc>  /      <NUM>
```

accentuate each NUMeral
--de +*TWEE* van Breda
--dan krijg je er +*TWEE*"

82. Acc_Superlatives:

```
<-acc>      →  <+acc>  /       "*ste"
                              NOT<VERB>
```

accentuate each superlative (either ADJective or ADVerb); these can be recognised by the suffix "st(e)" and a syntactic label which is NOT <VERB>.
--de *MOOISTE* koningin aller TIJDEN
--ik vind SNELLE AUTO'S het *LEUKSTE*"
--hij -*danste* de SAMBA

83. Acc_Between_ConjInt:

```
<-acc>      →    <+acc>      /       <CONJ>     ___          { IntGrens}
                                   {<CW,FW>}                 {ZinsGrens}
```

84. Acc_PrepPRON:

```
<PRON> <-acc>     →     <+acc>      /      <PREP>       ___
 ^64                                        ^55
```

sometimes, a PREP and personal PRONoun establish an adverbial PP (modifier). in these cases, the PP must be accentuated (on the PRON). feature ^55 indicates PREPositions starting adverbial PPs; feature ^64 indicates personal PRONs

85. Acc_LocPrep:

```
"b*en" <-acc>     →     <+acc>      /      "naar"        ___
<PREP>                                     <PREP>
```

accentuate the PP formed by ["naar" + locative PREP]; this PP functions as adverbial PP.
-- ik loop *naar +BINNEN*

86. Acc_EENS:

```
"eens" <-acc>     →     <+acc>      /      {"mee,niet,voor"}[1]                     ___    {"per,in"}[2]
                                           ( "met"            (NOT<PC>)0..n )[3]
```

87. Acc_EEN:

```
"een" <-acc>      →     <+acc>      /      "de"          ___
```

88. Acc_AL:

```
"al" <-acc> →     <+acc>      /     ___    <PRON>
                                            ^42
```

89. Acc_PRON:

```
<PRON> <-acc>      →     <+acc>       /      "wat"  ___   ^7
{^64,"dat"}
```

accentuate PRON if preceded by "wat" and followed by marked VERB.
--maar wat *DAT* betreft ...
--maar wat *MIJ* aangaat ...

MODULE SELECT_VERBACCS

90. Acc_VerbCluster:

```
( ( ({<PREP>})0..n      { <PREP>[1]      })  "te"[1]) <VERB>      (<Verb>)0..m      <VERB>
    {      }              NOT{"om,                  <CW>                          <FW>
                          door,zonder"} }
    {<SAT> }            { <SAT>          }
                          1                          2                               3
```

a 'verbal cluster' is accentuated with the priorities given in the specification. if the cluster contains a particle (PREP or SAT; PREP must be followed by "te"), then this particle (last in sequence of particles/prepositions) is accentuated (1). else, the routine searches for the first CW in the verbal cluster, and accentuates this VERB/CW (2). if there is no CW in the cluster, the last VERB/FW is accentuated (3).
this routine aborts if an accent has been given to any word within the verbal cluster: a cluster can contain maximally one accent. the last step (3) is optional, and can be controlled by an input parameter OBL in the call of this routine. if OBL=TRUE, then an accent within the cluster is obligatory, and the routine should put an accent on VERB/FWs if there are no particles or VERB/CWs. otherwise (OBL=FALSE), FWs are not to be accentuated, and if the cluster contains VERB/FWs exclusively, the cluster does not get any accent.
--het is lekker om de korst ## *OP* te eten #### (Prep)
--hij heeft het boek ## *TERUG* gegeven ####   (Sat)
--dat hij zijn afspraak ## *VERGETEN* heeft ### (CW)
--hij vroeg ons ## vandaag ## te *KOMEN* ####   (FW)

91. Adv_VerbAcc:

```
Acc_VerbCluster    /      <ADV> (<PC>)         ___
```

a verbal cluster is accentuated if it is preceded by an ADVerb; this is a modifier of the VERB cluster, not an accentuable argument.
OBL:=FALSE (see comments above)
--hij vroeg ons ## *VANDAAG* ## *DOOR te werken* ####

92. Intrans_VerbAcc:

```
Acc_VerbCluster    /      ZinsGrens    ( NOT<PC> )         (<PC>)       ___     {IntGrens }
                                       (NOT<PRON>)1..n                          {ZinsGrens}
```

a verbal cluster is accentuated if it stands between an I-initial Phi domain (X) and an IntGrens (i.e., the cluster is I-final). in those cases, the 'X' domain is probably the subject of the following (transitive) verb cluster, which is to be accentuated. this subject domain must be simple (no internal structure) and may not contain any PRONouns; in the latter case the 'X' domain could be the object of the following verbal cluster.
- OBL:=FALSE (see comments above).
--#### de *CHINESE STUDENTEN* ## *willen +UITGEVOERD zien* ### wat hen beloofd is ####"
--#### dat de man ## *zijn*<PRON> *MOEDER* ## *-op wilde bellen* ###

93. AccCues_VerbAcc:

```
Acc_VerbCluster    /      ^91    (<PC>)        ___
```

the words in the lefthand context are all labeled as UNDEF. nevertheless, they constitute (the final part of) an adverbial phrase (modifier), which triggers accentuation of the following verbal cluster.
OBL:=FALSE (see comments above).
--we hebben maanden *DOORGEWERKT*

94. NoAccInt_VerbAcc:

```
Acc_VerbCluster    /      {IntGrens}  (<-acc>)0..n      ___     (<-acc>)0..n      {IntGrens}
                          {ZinsGrens}                                             {ZinsGrens}
```

if the I-domain containing the verbal cluster contains no accents, then the verbal cluster is to be accentuated.
OBL:=FALSE (see comments above).
--#### je *+ZIET* ### dat hij *+GEKOMEN* is ####

95. AdvLocPP_VerbAcc:

```
{  Acc_VerbCluster    } /      {<PREP>}     (NOT PhiGrens)1..n      (PhiGrens)   ___   {IntGrens}
{(##){<PREP,SAT>}###*}           ^55                                                   {ZinsGrens}
    NOT"te,van"                { "in" }[1]  (ART) "<cap>*"[1]
```

some constituents are formally PPs, but used as an adverbial component (modifier) to the verbal cluster. in those cases, like in Adv_VerbAcc, the following cluster must be accentuated. this also includes cases where the preceding PP is a locative constituent, indicating the place of action.
--dat ik *volgens hem* ## *+GEZONDIGD* heb ####
--dat ik *in* Rome ## *+GEZONDIGD* heb ####
--dat ik *in* zijn ogen ## *-gezondigd* heb ####"
--ik bel Jan ## *na* donderdag ## *+OP* ####

96. BZ_PronObj_VerbAcc:

```
Acc_VerbCluster / NOT ZinsGrens     <PRON>      PhiGrens      ___     {IntGrens,ZinsGrens}
```

accentuate the verbal cluster in subordinate clauses ('BZ') if the argument/object of the verb (lefthand constituent) is a pronominal constituent. exclude sentence-initial PRONs, these may be subject constituents.
--dat ik *hem* *+GESPROKEN* heb ####
--#### *hij* *-spreekt* ### met niemand ####"

97. HZ_PronObj_VerbAcc:

```
Acc_VerbCluster   /     ___     PhiGrens    <PRON>      {IntGrens,ZinsGrens}
```

accentuate the verbal cluster in main clauses ('HZ') if the argument-or-object of the verb (righthand constituent) is a pronominal constituent.
--#### de man ## *+BEWEERT* ## *iets* ####"

98. KnownArg_VerbAcc:

```
Acc_VerbCluster
/      (Known_Constituent (##))[1]    ___    { { ZinsGrens,IntGrens }   }[1]
                                             { (##) Known_Constituent) }[2]
```

accentuate the verbal cluster if the object is 'known' (such object domains are deaccentuated in the following module). in effect, accent shifts from this object (NP) to the verbal cluster. this rule also correctly applies on subject NPs (Phi domains), if the sentence is intransitive [NP<known> + VP + ###*]. for 'Known_Constituent' see routine below.

--dat ik *zulke dingen* ## *+GEZEGD* heb ####   [1]
--de minister ## *+ONTKENDE* ## *deze geruchten* ####   [2]
--*deze machines* ## *+HAPEREN* ####

99. Known_Constituent:

```
Known_Constituent → TRUE
/      (NOT<PC>)n   (NOT<Verb>)n        {"die,dat"}<Pron>}     (NOT<Verb>)n      (NOT<PC>)n
                                        { ^21             }
                                        { ("zo'n")[1]     }     (NOT<Num>)[1]
```

- "*zulke* boekjes"   (TRUE)
  "*zo'n* soort"   (TRUE)
  "*zo'n drie* soorten"   (FALSE)

MODULE ADJ_ACCENTS:

100. DeAcc_KnownCWs:

```
<+acc>      →      <-acc>
/      (NOT<PC>)n       { Known_Constituent }        ( ___ )1..n      (NOT<PC>)n
                        { "*ere"            }        NOT<VERB>
                        { ^22               }        <CW>
```

deaccentuate all CWs (VERBs excluded) following a 'known-qualifier', until the following boundary. this routine mimicks the observation that 'given' information is not to be accentuated. cf. rule Known_Constituent above.

--dat hij *zulke boekjes* LEEST

101. DeAcc_Trailer:

```
<+acc>     →     <-acc>     /     IntGrens    ^84    ( ___ )1..n {ZinsGrens,IntGrens}
```

trailing sentences (starting with a word with feature ^84 "aldus,zo", etc.) are to be deaccentuated. the word itself is also deaccentuated.

--#### geld stinkt ### *aldus de bank* ####"

102. DeAcc_SecondLexAcc:

```
<ADV> <+acc>     →     <-acc>     /     <+ACC>      ___
^4^5                                    ^4^5
                                        <ADV,FW>
```

if two adjacent words <FW,ADV> have an accent (feature ^4) which is derived from the lexicon (feature ^5), then the second one must be deaccentuated.

-- ***+HELEMAAL*** *-niet*

103. DeAcc_Hangmat:

```
<+acc>     →     <-acc>     /     <CW>                (##)   ___        (##)   <CW>
*"ende"                           <+ACC>                                       <+ACC>
                                  NOT{<Num>,"*e"}                              NOT<Verb>
```

--een EVENWIJDIG *rijdende* GOEDERENTREIN

104. DeAcc_Epitheton:

```
^93 <+acc>    →     <-acc>     /     ___          (##)         "<cap>*"
<UNDEF><CW>
```

```
        deaccentuate epitheton (UNDEF^93), if followed by proper name.
        --#### premier LUBBERS ## HUICHELDE ###
```

105. DeAcc_Unit_after_NUM:

```
^95 <+acc>→<-acc>      /     { <NUM> }  ___   (PhiGrens)  { <CW> }
                             { ^13   }                    { ^52  }

        -- DRIE gulden TWINTIG
        -- ENKELE guldens van de HONDERD
```

106. DeAcc_Hangmat_NUM:

```
^11 <+acc>→<-acc>      /     <NUM> ___    <CW>
                                         NOT<VERB>

        -- DRIE miljoen SCHAPEN
```

107. DeAcc_Hangmat_NP:

```
<CW><+acc>→<-acc>      /     ^82   ___   <+acc>

        -- de ERG verweerde MUREN
```

APPENDIX 2.1 : FEATURES IN *PROS* LEXICON

0. VERB

^06	FW_Before_Te
^07	Wat_X_Betreft
^08	Direct_Verb

1. NUMERAL

^11	Numeral_A
^12	Numeral_B
^13	Numeral_C

2. QUALIFIER

^21	Known_Qual1
^22	Known_Qual2
^23	Universal_Qual
^24	Deacc_Enige
^25	Negative_Qual
^26	Zo_Dat_Acc
^27	Partitive_Qual
^28	Accent_Wel
^29	NoComp_Comp

3. COMPL

^31	Wh-Comp
^33	Dat_Comp
^34	Om_Comp
^35	Wat_Comp

4. ARTICLE

^41	After_DE
^42	After_AL
^43	Impers_Pron
^44	Definite_Art
^45	Initi_P_Expr

5. PREPosition

^50	Infinit_Mark
^51	NonPrep_Sat
^52	Partitive_Prep
^53	NonParticle_Prep
^54	EllipsZin_Prep
^55	AdvPP_Prep
^56	Direction_Prep
^57	Van_Prep
^58	Locative_Prep
^59	PostP_Expr

6. PRONOUN

^61	Poss_Pron
^62	Subject_Pron
^63	Person_Pron
^64	StrongObject_Pron
^65	Dat_Pron
^66	StrongSubject_Pron

7. CONJunctive

^71	Complex_Prep2
^72	Complex_Prep3
^73	Complex_Prep4
^74	Complex_Prep1
^75	EN_Conj
^76	MAAR_Conj
^77	Compl_Prep5

8. ADVERB

^81	[+R]_Adv
^82	Intensifier_Adv
^83	Comparative_Adv
^84	Trailer_Adv
^85	WEL_Adv
^86	EENS_Acc1
^87	EENS_Acc2
^88	Er_Pron
^89	Adverb_Prep

9. UNDEFINED

^91	Temporal_NP_Kern
^92	Quant_NP_Kern
^93	Epitheton_Noun
^94	Dummy_Noun
^95	Deacc_Qual_Noun
^96	MidPP_Noun
^97	Deacc_GANG
^98	Deacc_Def_Art

APPENDIX 2.2 : CONTENTS OF *PROS* LEXICON

Function words are printed in boldface; each word is followed by a 4-digit syntactic code (explained in section 4.3.2), optionally followed by additional lexical features (explained in section 4.3.2 and in Appendix 2.1).

a	5010	*51 53*
aan	5010	*56 59 74*
aangaande	5010	
aangaat	9002	*07*
aangezien	3010	
aanging	9002	*07*
aanleiding	9000	*96*
aantal	1002	*13*
aantallen	1002	*13*
aanzien	9000	*96*
ab	5010	*53*
acht	1001	*11*
achter	5010	
af	2010	*51 71 77*
afdeling	9000	
afloop	9000	*96*
afwachting	9000	*96*
al	8010	
aldus	8010	*84*
aleer	3010	
alhoewel	3010	
alle	1001	*23*
alleen	8000	
alles	6010	
als	3010	*45*
alsmede	7010	
alsof	3010	
alsook	7010	
althans	8010	
altijd	8000	
alvorens	3010	
ander	9001	*22*
andere	9001	*22*
anderen	9000	
anders	8000	
ante	5010	*53*
anti	5010	*53*
april	9000	*91*
attentie	9000	*96*
augustus	9000	*91*
avond	9000	*91 98*
beetje	8010	
behalve	5011	*53*
behoeve	9000	*96*
behoud	9000	*96*
behoudens	5011	*53*
behulp	9000	*96*
beide	1001	*22*
beiden	6000	
bekend	2010	
ben	0110	
beneden	5010	*53 77 89*
benevens	5011	*53*
bent	0110	
bepaalde	9000	
beschikking	9000	*96*
best	8000	
bestrijding	9000	*96*
beter	8000	
betreffende	5010	*53*
betreft	9002	*07*
betrekking	9000	*96*
betrof	9002	*07*
bevordering	9000	*96*
bezig	2010	
bezit	9000	*96*
bij	5010	*56*
bijna	8001	*25*
bijvoorbeeld	8010	
binnen	5010	*77 89*
bleef	0110	
bleek	0110	
bleken	0110	
bleven	0110	
blijf	0110	
blijft	0110	
blijk	0110	
blijkbaar	8010	
blijken	0310	
blijkens	5010	*53*
blijkt	0110	
blijven	0310	
bond	9000	
boven	5010	*77 89*
bovendien	8000	
buiten	5010	*77 89*
burgemeester	9000	*93*
buurt	9000	*99*
circa	5010	*53*
club	9000	
con	5010	*53*
contra	5010	*53*

daar	8010	*81*
daaraan	8010	
daarachter	8010	
daaraf	8010	
daarbij	8010	
daardoor	8010	
daarheen	8010	
daarin	8010	
daarmee	8010	
daarnaar	8010	
daarom	8010	
daaronder	8010	
daarop	8010	
daarover	8010	
daartegen	8010	
daartoe	8010	
daaruit	8010	
daarvan	8010	*52*
daarvoor	8010	
dag	9000	*91 93*
dagen	9000	*91*
dan	8010	*73*
dan	9010	*73*
dat	3010	*26 32*
dat	6010	*21*
datgene	6010	
de	4010	*42 44*
december	9000	*91*
deden	0110	
deed	0110	
deel	9000	*27*
degene	6010	
degenen	6010	
den	4010	
der	4010	*52*
dergelijk	9002	*21*
dergelijke	9002	*21*
dergelijks	9000	
derhalve	8010	
deze	6012	*21 42*
dezelfde	9001	*22*
dicht	9000	
die	3010	*42*
die	6010	*21*
diegene	6010	
dientengevolge	8000	
diezelfde	9001	*22*
dikwijls	8000	
ding	9002	*52*
dingen	9000	
directeur	9000	*93*
dit	6010	*21 42*
doch	7010	
doctor	9000	*93*
doe	0110	
doen	0310	
doet	0110	
dolgraag	8000	
dollar	9000	*95*
door	5010	*54 55 56*
doordat	3010	
drie	1001	*11*
duizend	1001	*11*
duizenden	1001	*12*
dus	8010	
echt	8001	*82*
echter	8010	*76*
een	1001	*11*
een	4010	
eenmaal	8010	
eens	8010	
eenzelfde	9001	*22*
eerst	8000	
eeuw	9000	*91*
eeuwen	9000	*91*
eigen	9000	
eigenlijk	8010	
elders	9001	*22*
elf	1001	*11*
elk	9001	*23*
elkaar	6010	
elkander	6010	
elke	9001	*23*
en	7010	*71 72 73*
enige	1000	
enkel	1000	*13*
enkele	1000	*13*
er	8010	*81*
eraan	8010	
erachter	8010	
eraf	8010	
erbij	8010	
erdoor	8010	
erg	8001	*82*
ergens	8010	*81*
erheen	8010	
erin	8010	
ermee	8010	
ernaar	8010	
erom	8010	
eronder	8010	
erop	8010	
erover	8010	
ertegen	8010	
ertoe	8010	
eruit	8010	
ervan	8010	*52*
ervoor	8010	
even	8010	
evenals	7010	

eveneens	8000	
evenmin	8001	*22*
eventueel	8010	
ex	5010	*53*
februari	9000	*91*
federatie	9000	
feit	9000	*94*
ga	0110	
gaan	0310	
gaat	0110	
gaf	0110	
gang	9000	*97*
gauw	8000	
gaven	0110	
ge	6010	
gebied	9000	
gebieden	9000	
gebleken	0210	
gebleven	0210	
gebrek	9000	*96*
gebruik	9000	*96 98*
gedaan	0210	
gedurende	5010	*53*
geef	0110	
geeft	0110	
geen	1001	*13 25 82*
gegaan	0210	
gegeven	0210	
gegooid	0210	*08*
gehaald	0210	*08*
gehad	0210	
geheel	9000	
gehouden	0210	
gekeken	0200	*09*
gekomen	0210	
gekregen	0210	
gekund	0210	
geleden	8010	
gelegd	0210	*08*
gelegen	0210	
gelegenheid	9000	*96*
geleken	0210	
gemaakt	0210	
gemeente	9000	
gemis	9000	*96*
gemoeten	0210	
gemogen	0210	
genomen	0210	
gereed	2010	
gestaan	0210	
geval	9002	*92*
gevallen	9002	*92*
geven	0310	
gevolg	9000	*96*
gevonden	0210	
gevraagd	0200	*09*
geweest	0210	
geweten	0210	
gewild	0210	
gewoon	8010	
geworden	0210	
geworpen	0210	*08*
gezegd	0200	
gezet	0210	*08*
gezeten	0210	
gezien	0210	
gezien	5010	*53*
gij	6010	
ging	0110	
gingen	0110	
gisteren	9000	*91*
goed	8000	
gooi	0110	*08*
gooide	0110	*08*
gooiden	0110	*08*
gooien	0310	*08*
gooit	0110	*08*
graaf	9000	*93*
graag	8000	
graden	9000	*95*
gram	9000	*95*
gravin	9000	*93*
gros	9001	*27*
gulden	9000	*95*
haal	0110	*08*
haalde	0110	*08*
haalden	0110	*08*
haalt	0110	*08*
haar	6010	*61 63 64*
had	0110	
hadden	0110	
halen	0310	*08*
half	1001	*13*
halfweg	5010	*53*
halverwege	5010	*53*
hangend	5010	*53*
hangende	5010	*53*
heb	0110	
hebben	0310	
hebt	0110	
heeft	0110	
heel	8001	*82*
heen	2010	
heer	9000	
helaas	8000	
helemaal	8000	
helft	9001	*27*
hem	6010	*63 64*
hen	6010	*63 64*
heren	9000	

het	4010	*42 44*
het	6010	*43*
hetgeen	3010	
hetwelk	3010	
hetzij	3010	
hield	0110	
hielden	0110	
hier	8010	*81*
hieraan	8010	
hierachter	8010	
hieraf	8010	
hierbij	8010	
hierdoor	8010	
hierheen	8010	
hierin	8010	
hiermee	8010	
hiernaar	8010	
hierom	8010	
hieronder	8010	
hierop	8010	
hierover	8010	
hiertegen	8010	
hiertoe	8010	
hieruit	8010	
hiervan	8010	
hiervoor	8010	
hij	6010	*62 63 66*
hoe	3010	
hoedanig	3010	
hoef	0110	
hoefde	0110	
hoefden	0110	
hoeft	0110	
hoeveel	3010	
hoeveelheden	9000	
hoeveelheid	9000	
hoeven	0310	
hoewel	3010	
hoezeer	3010	
honderd	1001	*11*
honderden	1001	*12*
hoogte	9000	*96*
hoop	9000	*96*
hopelijk	8000	
houd	0110	
houden	0310	
houdt	0110	
huidig	9001	*22*
huidige	9001	*22*
hun	6010	*61*
ie	6010	*63*
ieder	9001	*23*
iedere	9001	*23*
iedereen	6010	
iemand	6010	
iets	6010	
ik	6010	*62 63 66*
immers	8010	
in	5010	*56 58*
indien	3010	
ineens	8000	
ingaande	5010	*53*
ingang	9000	*96*
ingeval	3010	
ingevolge	5010	*53*
inmiddels	8010	
instantie	9002	*92*
inter	5010	*53*
intra	5010	*53*
intussen	8010	
inzake	5010	*53*
is	0110	
jaar	9000	*91 95*
januari	9000	*91*
jaren	9000	*91 95*
je	6010	*62 63*
jegens	5010	*53*
jij	6010	*62 63 66*
jou	6010	*63 64*
jouw	6010	*61*
juist	8000	
juli	9000	*91*
jullie	6010	*62 63 64 66*
juni	9000	*91*
kan	0110	
keek	0100	*09*
keer	9002	*91*
keken	0100	*09*
kennis	9000	*96*
kijk	0100	*09*
kijken	0300	*09*
kijkt	0100	*09*
kilo	9000	*92 95*
kilometer	9000	*95*
klaar	2010	
klasse	9000	
kom	0110	
komen	0310	
komt	0110	
kon	0110	
konden	0110	
koning	9000	*93*
koningin	9000	*93*
krachtens	5010	*53*
kreeg	0110	
kregen	0110	
krijg	0110	
krijgen	0310	
krijgt	0110	
kun	0110	

kunnen	0310	
kunt	0110	
kwam	0110	
kwamen	0110	
kwart	9001	*27*
laat	0110	
lag	0110	
lagen	0110	
land	9000	
lang	8000	
langs	5010	*53*
laten	0310	
leden	0100	
leden	9000	
leek	0110	
leg	0110	*08*
legde	0110	*08*
legden	0110	*08*
leggen	0310	*08*
legt	0110	*08*
leiding	9000	*96*
leken	0110	
lid	9000	
liet	0110	
lieten	0110	
lig	0110	
liggen	0310	
ligt	0110	
lijk	0110	
lijken	0310	
lijkt	0110	
los	2010	
m'n	6010	*61*
maak	0110	
maakt	0110	
maakte	0110	
maakten	0110	
maal	9002	*91*
maand	9000	*91*
maanden	9000	*91*
maar	7010	
maar	8010	
maart	9000	*91*
maatschappij	9000	
mag	0110	
maken	0310	
man	9000	*98*
manier	9002	*92*
manieren	9002	*92*
mate	8000	
me	6010	*63*
mede	8000	
medewerking	9000	*96*
mee	2010	*51 86*
meer	8001	*82*
meestal	8000	
meeste	9001	*21 27*
mei	9000	*91*
mekaar	6010	
men	6010	*62 63*
meneer	9000	
menig	9001	*21*
menige	9001	*21*
menigeen	6010	
mensen	9000	
met	5010	
	53 55 59 72 87	
meteen	8000	
meter	9000	*95*
mevrouw	9000	
middag	9000	*91*
middel	9000	*96*
midden	9010	
mij	6010	*63 64*
mijn	6010	*61*
miljard	1001	*11*
miljarden	1001	*12*
miljoen	1001	*11*
miljoenen	1001	*12*
minder	8001	*82*
minister	9000	*93 98*
ministerie	9000	
minstens	8001	*25*
minuten	9000	*95*
misschien	8010	
mits	3010	
mocht	0110	
mochten	0110	
moest	0110	
moesten	0110	
moet	0110	
moeten	0310	
mogen	0310	
morgen	9000	*91*
na	5010	*55*
naam	9000	*96*
naar	5010	*53 56*
naardien	8010	
naarmate	3010	
naast	5010	*53 56*
nabij	5010	
nacht	9000	*91*
nadat	3010	
nagedachtenis	9000	*96*
nam	0110	
name	9000	*96*
namelijk	8010	
namen	0110	
namens	5010	*53*
natuurlijk	8010	

nauwelijks	8001	*25*
neem	0110	
neemt	0110	
neer	2010	
negen	1001	*11*
nemen	0310	
nergens	8011	*81*
net	8001	*25*
neven	7010	
nevens	5010	*53*
niemand	6010	
niet	8001	*25 86*
niets	6010	
niks	6010	
noch	7010	
nochtans	8010	
nodig	2010	
nog	8010	
nogal	8010	
nooit	8000	
nopens	5010	*53*
nou	8010	
november	9000	*91*
nu	8010	*73*
nummer	9000	
o.a.	8010	
ochtend	9000	*91*
of	3010	*32*
of	7010	
ofschoon	3010	
oktober	9000	*91*
om	5010	*53 54*
omdat	3010	
omstreeks	5010	*53*
omtrent	5010	*53*
ondanks	5011	*53 55*
onder	5010	
ondertussen	8010	
ongeveer	8012	*25*
onlangs	8000	
ons	6010	*63 64*
onze	6010	*61*
ooit	8010	
ook	8000	
op	5010	*56*
opdat	3010	
opeens	8000	
open	9000	
opnieuw	8000	
over	5010	*56*
overal	8011	*81*
overeenkomstig	5010	*53*
overeenstemming	9000	*96*
overige	9001	*22*
overigens	8010	

overstaan	9000	*96*
paar	1002	*13*
pas	8010	
per	5010	*52 53 55 58*
personen	9000	
plaats	2010	
plaats	9000	*96 98*
premier	9000	*93 98*
president	9000	*93*
prins	9000	*93*
prinses	9000	*93*
procent	9000	*92 95*
professor	9000	*93*
provincie	9000	*98*
raad	9000	
reeds	8010	
reeks	1002	*13*
rest	9001	*27*
rond	5010	
rondom	5010	*53*
ruim	8001	*25*
's	8010	
samen	8001	*25*
samenwerking	9000	*96*
seconden	9000	*95*
sedert	5010	*53*
september	9000	*91*
serie	1002	*13*
sinds	5010	*53 55*
sint	9000	*93*
slechts	8010	*25*
sommige	9000	
soms	8010	
soort	9000	
soorten	9000	
sta	0110	
staan	0310	
staat	0110	
stad	9000	*98*
steeds	8000	
stichting	9000	
stond	0110	
stonden	0110	
straks	8010	
strijd	9000	*96*
't	4010	
te	9010	*42 51 53*
tegen	5010	
tegenover	5010	*53*
telken	4010	
telkens	8000	
telker	4010	
ten	4010	*45*
teneinde	3010	
tenminste	8001	*25*

tenslotte	8010	
tenzij	3010	
ter	4010	*45*
terecht	2010	
terug	2010	
terwijl	3010	
tevens	8000	
tevoren	8000	
tevreden	9001	
tezamen	8001	*25*
thans	8010	
tien	1001	*11*
tientallen	1001	*12*
tijdens	5010	*53 55*
toch	8000	
toe	2010	*51 71 74*
toen	3010	
toen	8010	
ton	9000	*95*
tot	5010	*53 59 72*
totdat	3010	
trouwens	8010	
tussen	5010	*53*
twaalf	1001	*11*
twee	1001	*11*
type	9000	
u	6010	
uit	5010	*56 77*
uitgezonderd	5011	*53*
uren	9000	*91 95*
uur	9000	*91 95*
uw	6010	*61*
vaak	8000	
valt	0110	
van	5010	
	52 53 56 59 77 97	
vanachter	5010	*53*
vanaf	5010	*53*
vandaan	2010	
vanonder	5010	*53*
vanuit	5010	*53*
vanwege	5010	*53*
vast	2010	
veel	8001	*82*
vele	1001	*13*
ver	8000	
verband	9000	*96*
verdachte	9000	*98*
vereniging	9000	
vermits	3010	
versus	5010	*53*
vervanging	9000	*96*
vervolgens	8010	
via	5010	*53 55*
viel	0110	
vier	1001	*11*
vijf	1001	*11*
vind	0110	
vinden	0310	
vindt	0110	
volgend	9001	*22*
volgende	9001	*22*
volgens	5010	*53 55*
volstrekt	8000	
vond	0110	
vonden	0110	
voor	5010	*86*
vooral	8000	
vooraleer	8010	
voordat	3010	
voort	2010	
voorzover	3010	
vorig	9001	*22*
vorige	9001	*22*
vraag	0100	*09*
vraag	9000	*09 94*
vraagt	0100	*09*
vragen	0300	*09*
vragen	9000	*09*
vrij	2010	
vrijwel	8010	
vroeg	0100	*09*
vroeg	8000	
vroegen	0100	*09*
vroeger	8000	
vrouw	9000	*98*
waar	3010	*81*
waaraan	3010	
waarbij	3010	
waarbinnen	3010	
waarboven	3010	
waarbuiten	3010	
waarde	9000	*96*
waardoor	3010	
waarin	3010	
waarlangs	3010	
waarmee	3010	
waarna	3010	
waarnaar	3010	
waarnaast	3010	
waarnevens	3010	
waarom	3010	
waaromtrent	3010	
waaronder	3010	
waaronderdoor	3010	
waarop	3010	
waarover	3010	
waaroverheen	3010	
waartegen	3010	
waartegenover	3010	

waartoe	3010	
waartussen	3010	
waaruit	3010	
waarvan	3010	
waarvoor	3010	
waarzonder	3010	
wanneer	3010	
want	7010	
waren	0110	
was	0110	
wat	3010	*31*
wat	6010	*35*
we	6010	*62 63*
weer	8010	
weet	0100	*06*
weg	2010	
wegens	5010	*53 55*
wel	8010	
weliswaar	8010	
welk	3011	*31*
welke	3011	*31*
wellicht	8010	
werd	0110	
werden	0110	
werp	0110	*08*
werpen	0310	*08*
werpt	0110	*08*
weten	0300	*06*
wethouder	9000	*93*
wie	3010	*31*
wierp	0110	*08*
wierpen	0110	*08*
wij	6010	*62 63 66*
wijl	3010	
wijze	9002	*92 96*
wil	0110	
wilde	0110	
wilden	0110	
willen	0310	
wilt	0110	
wist	0100	*06*
wisten	0100	*06*
word	0110	
worden	0310	
wordt	0110	
z'n	6010	*61*
zaak	9000	*98*
zag	0100	*06*
zagen	0100	*06*
zake	9000	*96*
zaken	9000	*97*
zal	0110	
zat	0110	
zaten	0110	
ze	6010	*62 63*
zeer	8001	*82*
zeg	0100	
zeggen	0300	*84*
zegt	0100	*84*
zei	0100	*84*
zeiden	0100	*84*
zelden	8000	
zelf	9000	
zelfde	9001	*22*
zelfs	8001	*28*
zelve	9000	
zes	1001	*11*
zet	0110	*08*
zette	0110	*08*
zetten	0310	*08*
zeven	1001	*11*
zich	6010	*63*
zichzelf	6010	
zie	0110	
zien	0300	*06*
ziet	0100	*06*
zij	6010	*62 63 66*
zijn	0310	
zijn	6010	*61*
zit	0110	
zitten	0310	
zo	8010	*26 84*
zo'n	9010	*21 25 26*
zoals	3010	
zodat	3010	
zodra	3010	
zogeheten	9000	
zogenaamd	9000	
zogenaamde	9000	
zoiets	6010	
zolang	3010	
zomin	3010	
zonder	5011	*53 54 55*
zou	0110	
zouden	0110	
zover	8010	
zowat	8010	
zowel	8010	
zul	0110	
zulk	9002	*21*
zulke	9002	*21*
zulks	6010	
zullen	0310	
zult	0110	