(ARIPUC 23, 1981 p. 119-152)

SYNTAX, MORPHOLOGY, AND PHONOLOGY IN TEXT-TO-SPEECH SYSTEMS

Peter Molbæk Hansen

The paper is concerned with the integration of linguistic information in text-to-speech systems. Research in synthesis proper is at a stage where the need for systematic integration of comprehensive linguistic information in such systems is making itself felt more than ever. A surface structure parsing system is presented whose main virtue is that it permits linguists to express syntactic as well as lexical and morphological regularities and irregularities of a language in a simple and easy-to-learn formalism. Most aspects of the system are seen in the light of Danish and - sporadically - English and Finnish surface structure.

I. INTRODUCTION

In recent years there has been considerable progress in the design of automatic text-to-speech systems (henceforth TTS-systems) for many languages. The development of advanced techniques and tools for generating high-quality synthetic speech signals has gradually entailed a shift of focus in speech synthesis research from technological to phonetic aspects.

At the linguistic end of TTS-systems there has, however, been little emphasis on the development of *general* tools and formalisms, and the exploitation of insights from computational linguistics has hitherto been sporadic. All TTS-systems are faced with the problem of supplying the synthesis component with sufficient phonetic information, typically in the form of phonetic transcriptions derived from text, but there has been a tendency to use rather diverse algorithms relying heavily on language specific peculiarities instead of using formalisms and parser algorithms of a more general nature. Incidentally, in most older systems syntactic and morphological information is not exploited at all (Carlson & Granström

1975), in other systems morphological and lexical information is exploited but not combined with syntactic information (Molbæk Hansen 1983). In some of the best systems, lexical as well as morphological and syntactic information is integrated, but morphology and syntax appear as distinct components, each with its own structure and algorithm (Allen et al. 1987, p. 23ff).

As the acoustic quality of synthetic speech as such becomes comparable to that of natural speech, the need for higher level linguistic information of all kinds relevant to pronunciation increases, and it is therefore important to develop formalisms which permit linguists to express lexical, morphological, and syntactic structuring in linguistically meaningful ways, and to develop parsing systems which can cope with information expressed in such formalisms in an efficient way.

The major part of the present paper is the presentation of a *set of conventions for declaring linguistic structures* of various kinds in a *linguist-oriented* way: the declarative conventions permit the linguist to formulate lexical (including morphophonemic), morphological, and syntactic structuring in a language independent formalism which is easy to learn. The system is called SSPS (surface structure parsing system), and its main components are a *lexicon system*, a *constituent structure grammar*, and a *chart-based parser*. In SSPS no formal distinction is made between syntax and morphology: surface structures are seen as tree structures - deep of flat as the case may be - which can be described by a set of rewrite rules, i.e. a production system, whose terminal symbols are morphemes and whose root symbol may be any category which the linguist wishes to consider, e.g. STEM, WORD, or SENTENCE. The system includes a parser, which "understands" the declarations of the formalism and interprets them as a set of instructions for analyzing orthographic input and for transforming it to another format, e.g. a morphophonemic representation.

In Section II the basic declarative conventions of SSPS are introduced, the linguistic phenomena which motivate them are illustrated, and the system is classified typologically in relation to other formalisms. After this introduction the individual components of SSPS are described in detail.

In section III the use of SSPS in a TTS-system for Danish is illustrated. In particular, the use of *morphosyntactic features* to reduce overgeneration in both syntax and morphology is exemplified.

In section IV the SSPS parser is presented in outline, and I conclude the paper in section V with a brief personal comment on the possibilities of harmonizing the phonological components with the linguistic components in TTS-systems.

II. THE SSPS FORMALISM

A. Basic Properties

The core of the formalism is a *constituent structure grammar* describing what one might call "categorial surface structures". By this term I refer to surface structures viewed as arrangements of traditional, structurally motivated categories labelled word, root, stem, affix, etc.

An extremely simple grammar of this type - describing only morphological structure - might look like (1)

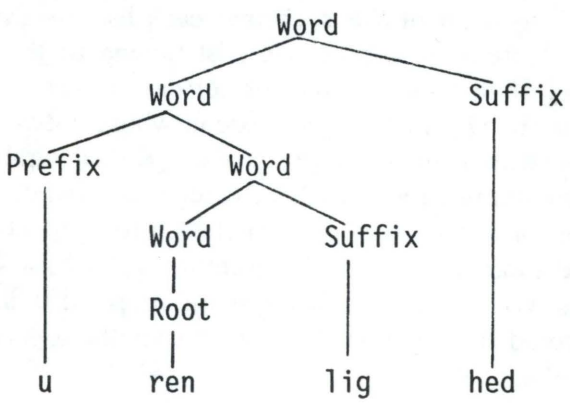
(1)

```
Word   ->   Root
Word   ->   Word Suffix
Word   ->   Prefix Word
Root   ->   ren (clean)
Prefix ->   u   (un-)
Suffix ->   lig (-ly)
Suffix ->   hed (-ness)
```

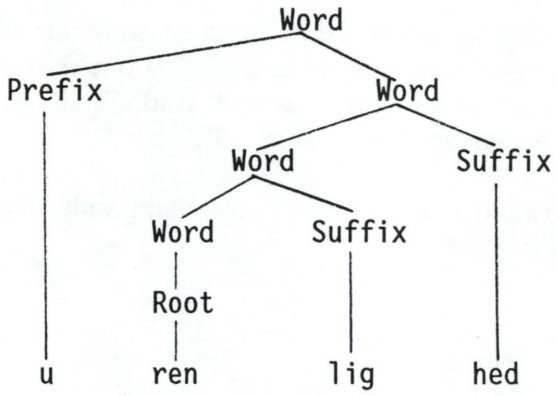
The grammar (1) has the well known formal properties of a *context free grammar*, in this case one including *recursive rules.* Such a grammar is to all intents and purposes powerfull enough to accomodate any structural type one may want to operate with in morphology and *surface* syntax.

As can easily be seen, however, the particular grammar (1) *overgenerates.* In addition to generating (or accepting) the word *urenlighed* "uncleanliness", assigning to it the structure (2), which is the natural one for this word, it will assign several other structures to it, for instance (3), thus coming out with several distinct "solutions".

(2)



(3)



Moreover, (1) will generate and accept incorrect word forms like *uuuurenligligheđ.* Clearly (1) is too permissive. On the other hand, since (2) can in fact be defended as a "correct" structural description of *urenlighed,* the recursive constituent structure grammar seems to express at least some morphological properties of Danish words in a satisfactory way, and thus should not be dismissed off hand. What is needed, of course, is some systematic way of expressing *restrictions* in the combinability of constituents.

As is well known, grammars like (1) usually leave out rewrite rules whose right side consists of a single terminal symbol (the four lower rules in (1)). Instead the *preterminal symbols*, i.e. the symbols on the left side of the rewrite symbol in rules of the latter kind, appear formally as the terminal symbols of the grammar, and any such symbol is supposed to represent an individual lexical item belonging to the category designated by that symbol. In other words, the grammar presupposes the existence of a *lexicon* whose items are marked off as belonging to one or more categories. Technically, such a lexicon can be arranged in at least two basic ways: 1. as a simple list of items each of which has one or more categorial labels, or 2. as a set of *lists* such that each list has a categorial label and such that all items in a particular list belong to the category identified by the label of that list. In the former case a terminal symbol in the grammar refers to any item in the lexicon whose categorial label corresponds with the symbol. In the latter case a terminal symbol in the grammar refers to any item of the list whose categorial label corresponds with the symbol. The former strategy is often chosen for syntactic parsing systems where the terminal symbols of the grammar refer to *word classes* like nouns, adjectives, verbs, etc. In such systems a lexical item like the English word *drink* would appear in the lexicon as something like this:

drink noun, verb

In SSPS the latter strategy has been adopted: The lexicon is partitioned into separate lists with labels of the type *prefixes*, *roots*, *suffixes*, *endings*, etc., i.e. labels referring to *distributionally defined morpheme types*, and a terminal symbol in the grammar refers to any item from lists having the symbol as its label. Thus, a rule like

```
STEM  ->  pref root
```

presupposes the existence of two lexicon partitions labelled 'pref' and 'root', respectively, and it says that a STEM may consist of an item from the former followed by an item from the latter. Since the terminal symbols of the grammar refer (indirectly) to *morphemes*, a traditional syntactic rule like

```
NP -> adj noun
```

where the terminal symbols are word classes, must be expressed in a different way in SSPS, where there is typically no lexical partitions labelled 'adj' or 'noun', since words are not in general coextensive with morphemes. If a linguist wishes to write an SSPS rule referring to a word class, he must use *features*. In several recent formalisms - see e.g. Karttunen (1986) and Whitelock (1988) - grammar symbols are not atomic as they are in the grammar (1) and in pure context free grammars. This is also the case in SSPS. Lexical entries have an internal structure comprising *a set of features* which may designate, among other things, such properties as word class, and the symbols in the grammar may refer to such features. In fact the above-mentioned rule would typically be translated into

NP -> WORD(?A) (?N)WORD

in an SSPS grammar for Danish. The contents of the parentheses express restrictions in the combinability of two consecutive constituents of the category WORD, namely restrictions referring to the feature compositions of the constituents. The technical details of these notational facilities will be described in section III.

The use of features does not mean that SSPS is formally stronger (in the sense of the Chomsky hierarchy) than a context free grammar: the grammar and the lexicon system could in principle be translated into a context free grammar with atomic symbols. But the advantages of relying on featured constituents are 1) that it is a natural way to express individual properties of morphemes, 2) that it is easy to modify algorithms for atomic context free parsing in such a way as to take feature restrictions into account, and 3) that such algorithms tend to be faster than parsers for atomic context free grammar-lexicon systems with equivalent strength.

The strategy of having terminal grammar symbols refer to distributionally defined morpheme types is a natural consequence of the fact that SSPS is designed to describe both morphology and surface syntax: roots, prefixes, etc. are the terminal constituents of words in much the same way as nouns, adjectives, etc. are the terminal constituents of surface sentences. The use of a single constituent structure grammar to cover both surface syntax and morphology is in accordance with - and partly inspired by - Selkirk's extended version of Chomsky's (1970) X-bar theory, cf. that Selkirk includes morphological constituents in the hierarchy of categorial types (Selkirk 1982, p. 6f). The design of SSPS is not, however, seriously committed to any specific linguistic theory.

In recent years Koskenniemi's (1983) *two-level morphology* has dominated theory and practice in computational analysis of *morphological* structure. I have argued elsewhere (Molbæk Hansen forthcoming) that this kind of analysis is not well suited to systems where the specific format of the output of the morphological component is important. In a TTS-system the output format is of course particularly important, because it is supposed to contain the phonological information in string form, more particularly as strings of morphophonemic segments and boundaries. As a consequence, the lexicon system of SSPS differs radically from that of two-level morphology, particularly in that the output strings are *entirely independent of the parser algorithm and of the rules describing orthographic alternation of morphemes*.

As the linguistic component of a TTS-system, the SSPS parser has three main tasks:
1) to *identify* input texts as sequences of morphemes in written form. In this connection orthographically alternating forms of the same morpheme must be taken into account, cf. e.g. that the morpheme {gammel} 'old' appears in two different orthographic shapes, *gammel* and *gaml*.
2) to *output* structures which contain sufficient relevant phonological information for the pronunciation of the text to be computed. This implies, among other things, the conversion of the string format of the *terminal* material, i.e. the matched morphemes, into a format which is phonetically interpretable.
3) to *confer* the identified morpheme strings with lexical and grammatic information in order to exclude incorrect analyses, such as ['man'ən 'dœˀʁ] *'the man door' as the interpretation of the input text *manden dør*, instead of the correct one: ['man'ən 'døˀʁ] 'the man dies'.

Of these tasks 3) is indisputably the most difficult one. Overgeneration, i.e. the assignment of several structures to the same input, is a problem for all parsing systems, especially for systems including morphological analysis, and it might be argued that at least derivational and compositional morphology represents an unnecessary complication for a TTS-system, since the use of a lexeme-based lexicon comprising traditional dictionary forms would eliminate most sources of overgeneration at the word level (such as the incorrect analyses *kul-tur* and *kult-ur* in addition

to the correct *kultur* 'culture'). This argument can not, of course, be rejected on the grounds that a dictionary-based, morphology-free TTS-system would need a very large dictionary, since neither memory limitations nor lexical search time would be prohibiting factors in the light of hardware and software facilities now available. But it can be rejected on the grounds that morphological knowledge as such is needed anyway, especially for the interpretation of unidentified input words such as neologisms and spontaneous formations of new compounds. In most languages the inventory of morphemes is more well-defined than the inventory of well-formed lexemes, and the morphological structure *per se* is often crucial for pronunciation. Reduction - ideally elimination - of overgeneration must be obtained by *integrating* as much linguistic knowledge as possible, not by *ignoring* such knowledge. SSPS represents a step in that direction, at least for TTS-systems.

B. The Lexicon System

Since the terminal symbols of the constituent structure grammar refer to distributionally defined morpheme types, the lexicon is subdivided into separate partitions, each comprising entries of a particular type. However, the actual inventory of *lexicon partitions* in an SSPS system tends to be slightly richer than suggested by the coarse description of the principles given in the introduction. Thus in the SSPS-based TTS-implementation for Danish there are several prefix lists, several root lists, etc. The main reason for this is that the basic morpheme types - in Danish as well as in e.g. English - form distinct classes with respect to their combinability within single words with other basic types: in general, prefixes of Latin or Greek origin do not combine with native roots and *vice versa*, and there are other combinatorial restrictions as well which can be most naturally expressed by lexicon partitioning. A few examples of these combinatorial restrictions will make this point clear. (In the examples 'Latin' stands for 'of Latin origin', etc., and 'native' stands for 'inherited from Old Danish or borrowed from Middle Low German')

Most Latin Prefixes must be followed by a Latin root, and most native prefixes must be followed by a native root: *absolution* 'absolution' and *afløsning* 'release', not **abløsning.* and **afsolution.*

Most Latin suffixes must succeed a Latin root or stem, and most native suffixes must succeed a native root or stem: *immunitet* 'immunity' and *dumhed* 'stupidity', (literally: 'dumb-ness'), not **dummitet* and **immunhed.* These correlations are somewhat asymmetric, though: **immunhed* seems (to me at least) less ill-formed than **dummitet.*

Many Latin roots do not occur without a Latin prefix: *restaurere* 'restore' vs. **staurere.*

Certain Latin suffixes, in particular *-ere*, may, however, succeed certain

native roots: *snedkerere* 'to do carpentering' (*snedker* = 'carpenter').

Certain native suffixes may, likewise, succeed Latin roots or stems: *antikvarisk* 'second-hand' (about purchase of books) and *abrubthed* 'abrubtness', cf. **immunhed* above, and cf. the English *-ness* which behaves similarly.

I do not intend to give an exhaustive treatment of these combinatorial restrictions here, but for a lexicon system relying on distributionally defined morpheme types such phenomena obviously appeal to a more fine-grained partitioning than a mere division into 'prefixes', 'roots', etc.

1. MORPHOGRAPHEMIC ALTERNATION

In addition to the division of the lexicon according to the combinatorial pattern of morpheme types, there may be a subdivision of the lexicon partitions according to the *morphographemic alternation pattern* of lexical items. Any parsing system whose input format is orthographic and whose terminal symbols are morphemes, must cope with the fact that many morphemes appear in contextually conditioned orthographic variants, cf. English *heavy - heavier, fit - fitting*. As far as Danish is concerned, roots exhibit three basic graphemic patterns: some roots show an alternation between single and double final consonants, cf. *kat - katten* 'cat - the cat'; others show an e - zero alternation before final *l, n* or *r*, cf. *konvertibel - konvertible* 'convertible' (common gender, singular, indefinite vs. plural or definite); most roots, however, are graphemically constant in all contexts, cf. *hus - huset* 'house - the house'. Likewise, certain Latin prefixes exhibit graphemic alternation (reflecting phonological processes (assimilations) in Latin): *inaugurere - immobil - irrelevant - illativ; adhærere - assimilere - allativ.*

In Koskenniemi's two-level morphology (cf. above) the elimination of such orthographic ("surface") variation is taken care of by a set of rules expressing the contextually determined correspondences between "lexical" strings and "surface strings" in a letter-by-letter fashion. In SSPS this job is done in quite a different way which will be described below; but the *information* on the alternation patterns is linked with a subdivision of the lexicon partitions. In the Danish SSPS-system, for instance, there is a lexicon partition labelled rn which contains native roots. This lexicon partition is subdivided into four groups: rnrr, whose items exhibit no alternation (*hus - huse*), rnrd, whose items exhibit alternation between single and geminate final consonant (*kat - katten*), rnrsr, whose items exhibit simple e - zero alternation before final *l, n* or *r* (fængsel - fængsler), and rnrsd, whose items exhibit geminate consonant + e - single consonant + zero alternation before final *l, n* or *r* (gammel - gamle).

Since SSPS is a *declarative* system, the main partitioning as well as the subdivision according to graphemic alternation patterns and the exact

nature of each alternation pattern must be *declared* explicitly to the system. This is done by writing lines in a *lexicon declaration text* according to a set of naming conventions. A few examples - rather than extensive prose - will make these conventions and their meaning clear. In order to inform the system of the existence of the above-mentioned lexicon partitions containing native Danish roots, we simply write the following lines in the lexicon declaration text:

```
LEX rnrr
LEX rnrd
LEX rnrsr
LEX rnrsd
```

These declarations tell SSPS that there exist four lexicon partitions and that the terminal grammar symbols rnrr, rnrd, rnrsr, and rnrsd will match items from the corresponding lexicon partition.

Although I am concerned with the lexicon here, it may be expedient at this point to mention an important convention concerning the use of terminal symbols in grammar rules, a convention which is closely linked with the lexical naming conventions: *Any terminal symbol in a grammar rule will refer to lexical items from any concrete lexicon partition whose name begins with the symbol.* In the Danish application of SSPS four other concrete root lexicon partitions are declared (and exist), namely rfrr, rfrd, rfrsr, and rfrsd:

```
LEX rfrr
LEX rfrd
LEX rfrsr
LEX rfrsd
```

containing roots of foreign (Latin and Greek) origin. The convention just mentioned means that the symbol r in a grammar rule will refer to any item from these eight lexicon partitions (since their names all begin with r); the symbol rf and the symbol rfr will refer to any item from the four latter lexicon partitions; the four-letter symbol rfrsd, on the other hand, will only refer to any item from the concrete lexicon partition rfrsd. This naming convention enables the user to chose whatever degree of concreteness he sees fit when formulating particular grammar rules containing terminal symbols, i.e. rules referring to lexical items: since the alternation pattern of items from e.g. a particular root type is typically irrelevant in connection with the formulation of a rewrite rule referring to items of the distributionally defined type in question, the linguist should not be forced to worry about such matters when writing such a rule.

On the other hand, the declarations of the lexicon partitions rnrr etc. only

inform the system of the *existence* of such concrete lexicons, and a parser confronted with an SSPS grammar and orthographic input must of course cope with orthographic alternation, so the alternation patterns must be declared to the system somehow. In two-level morphology this declaration is taken care of by rules referring to strings of pairs of lexical and surface (orthographic) characters. In SSPS the alternation patterns are linked to *lexicon partitions*. When a concrete lexicon partition has been declared in the way just mentioned, the system will assume, unless otherwise informed, that its items exhibit no graphemic alternation. Thus, the above-mentioned concrete lexicon partition rnrr, which contains non-alternating roots, needs no further declaration. But the alternation pattern of items which do alternate is declared in a particular *alternation specification text* with a syntax of its own.

This text may start with a number of lines beginning with DEF, i.e. lines defining *classes*, e.g.

```
DEF V "aeuioyæøå"
```

which declares that the symbol V in the remaining lines of the declaration text stands for any of the characters a e u i o y æ ø å.

The alternation specifications proper are declared in lines beginning with TYP. Lines of this kind express the alternation patterns of the items of certain concrete lexicon partitions. Each such line is a series of *fields*. The first field is an *identification string* which should be identical with the *final* part of the label of some lexicon partition for which the user wants to declare a particular alternation pattern: Thus, for each of the concrete lexicon partitions whose labels end in d, sr, and sd in the Danish system there is a line whose first field is the identifying string. The next fields are abstract, symbolic expressions designating a. *the identificational shape* of the items in the concrete lexicon partitions, i.e. the shape in which they appear in their concrete lexicon partition, b. *the other shapes* in which the items appear, and c. *the contexts* in which the alternants occur.

Four type definition lines and four alternation specification lines are given in (4). The last four lines in (4) describe the behaviour of items from lexicon partitions with names ending in d, from lexicon partitions with names ending in sr, from lexicon partitions with names ending in sd, and from lexicon partitions with names ending in w. (Items from the latter partitions do not alternate themselves, but their orthographic shape is relevant to the alternation pattern of preceding morphemes, and this must be declared explicitly.)

(4)

```
DEF V "aeuioyæøå"
DEF C "rtpsdfgklbnm"
DEF L "rln"
DEF W "ei"
TYP d   @I0:VC>,@M:<!W  @I1:VC=C=>,@G:>W,@M:VC=C=<
TYP sr  @I0:CL>,@G:>W,@M:CL<  @I1:CeL>,@M:<!W
TYP sd  @I0:VC=C=L>,@G:>W,@M:VC=C=<  @I1:C=C=eL>,@M:<!W
TYP w   @G:@M
```

The meanings of the *keyword symbols* appearing in these lines i.e. the symbols beginning with @ and the symbol , (comma) are:

@I0: announces the alternant found in the physical lexicon.

@I1:, @I2: etc. announce other alternants.

@G: announces a graphemic condition which must be satisfied for the alternant to be legal and which is statable on the basis of the alternant in question.

@M: announces a graphemic condition which must be satisfied for the alternant to be legal and which is statable on the basis of the alternant in question *plus* additional information based on some other part of the word in question.

, is a separator between the description of an alternant and the description of the corresponding structural condition.

The morphographemic relations themselves are declared by writing *structural descriptions* of the alternants and of their contextual conditions. A structural description is a string of a) *class symbols* representing the classes defined in the DEF lines, b) *concrete symbols*, i.e. lower-case letters representing concrete letters of orthographic strings, and c) *one or both of the symbols* < *and* > representing the left and right boundary of morphemes in an orthographic string. Each class symbol in a structural description may be *indexed* by the symbol = which designates identity, e.g. if C= occurs in a line, then all C=’s in that line refer to the same consonant.

Each class symbol (whether indexed or not), each concrete symbol, and each *parenthesized string of such symbols* is a *substructure* which may be *followed* by one of the symbols ?, +, and * designating ’zero or one occurrences’, ’one or more occurrences’, and ’zero or more occurrences’ of the substructure, respectively, and each substructure may be *preceded* by the symbol ! which designates negation (complementation) of the

strings represented by the substructure.

After this brief presentation of the formal declarative structure - a variety of regular expressions - of the alternation specification text, let us translate the lines whose first fields are the strings d and w, respectively, into normal prose, in order to make clear what these lines actually tell the system.

The line

```
TYP d   @I0:VC>,@M:<!W          @I1:VC=C=>,@G:>W,@M:VC=C=<
```

may be translated thus:

"Items from concrete lexicon partitions whose names end in d appear in the concrete lexicon partition as strings ending in a vowel belonging to the defined class V followed by a single consonant belonging to the defined class C (@I0:VC>); this alternant occurs in orthographic words on condition that some following morpheme to be checked later in the word begins with a letter that does not belong to the defined class W (@M:<!W). Such items also appear as strings ending in a vowel followed by two identical consonants (@I1:VC=C=>); this alternant is *only legal* if it is followed to the right by a letter belonging to the defined class W (@G:>W) *and* on condition that some following morpheme to be checked later in the input *is preceded by* a vowel followed by two identical consonants (@M:VC=C=<)."

The line

TYP w @G:@M

may be translated thus:

"Items from concrete lexicon partitions whose names end in w do not exhibit alternation. (This is the default assumption when no @I0, @I1, etc. are mentioned.) Such items are only legal if a condition based on earlier parts of the input (@M:) is satisfied."

The difference between the meaning of the symbols @M: an @G: should be noted: @M: expresses the fact that certain combinability restrictions depend on morphographemic factors not deducible from the knowledge of the alternation pattern of a *single* morpheme, whereas @G: expresses the fact that other combinability restrictions are uniquely determinable by such knowledge. To spell out the two examples given above: in roots exhibiting alternation between single and geminate final consonant it may be safely stated that the alternant with a final geminate can only occur before shwa-initial suffixes and endings, and before the (native) suffixes

-ig, *-isk*, and *-ing*, i.e. before *orthographic e* and *i*. This does not mean, however, that the alternant with final single consonant is *excluded* before orthographic *e* and *i*; it may actually occur before these vowels if it is followed by another root in compounds, cf. *skakentusiast* 'enthusiastic chessplayer', literally 'chessenthusiast', and *glasindustri* 'glass industry'. Therefore such alternants can only be rejected if the *e* or the *i* turns out to be initial vowels in items from lexicon partitions of the w-type mentioned in (4) (shwa- or i-initial endings and suffixes).

Such facilities make it possible to state most alternation patterns in most languages and to link them with concrete lexicon partitions. In an SSPS implementation for Finnish, for instance, the inflectional and derivational suffixes exhibiting vowel harmony would be placed in a lexicon partition with an appropriate alternation identifier, say vh, as the final part of its label, and rules of the kind shown in (4) would be set up to express the alternation pattern characterising items from that lexicon partition.

In order to give this claim substance, I will show how the vowel harmony rules for Finnish set up by Koskenniemi (1983, p. 76) would be "translated" to the SSPS formalism. The suffixes exhibiting vowel harmony would be placed in a concrete lexicon partition declared in the lexicon declaration text as, say

LEX sfvh

and there would be a section in the alternation specification text looking like this:

(5)

```
DEF Hm "aouääöy"
DEF Vnb "ääöyie"
DEF Vf "ääöy"
DEF Vb "aou"

TYP vh @I0=<!Hm*Vf,@G:Vnb!Hm*< I1=<!Hm*Vb,@G:Vb!Hm*<
```

The latter specification says that items from lexicon partitions whose label end in vh have a lexical alternant which begins with zero or more letters not belonging to the defined class Hm (the segments which are neutral in relation to vowel harmony) followed by a front vowel (@I0:<!Hm*Vf); this alternant is only legal in the input if it is preceded by a member of the defined class Vnb followed by zero or more letters not belonging to the defined class Hm (@G:Vnb!Hm*<). Such items also appear as strings which begin with zero or more letters not belonging to the defined class Hm followed by a back vowel (@I1:<!Hm*Vb); this alternant is only legal in the input if it is preceded by a member of the defined class

Vnb followed by zero or more letters not belonging to the defined class Hm (@G:Vnb!Hm*<).

These examples should demonstrate that the structural description of graphemic alternation patterns may be *declared* in a general and reasonably simple language independent format.

Thanks to the formalism the linguist need not worry about how a parser program handles the information, but it may be mentioned that a parser which "understands" these conventions can be so constructed as to avoid superfluous lexical searching in cases where the declarations mention the @G: condition: thus in the analysis of an input word like *anklage* 'accuse' the Danish SSPS parser will never try to match the first four letters with items from the lexicon partition rnrsr (because the @G: condition of the sr-line in (4) will tell it that these letters should have been followed by an *e* in order for a search in that lexicon partition to be successful if the item ends in consonant + *l*). If the parser had not exploited this information it would have looked for a match in that lexicon partition, it would have found that these letters actually match the item *ankel* 'ankle' whose lexical alternant is *ankl*, and a hypothesis to the effect that this item is a correct identification of the first part of the word would have been set up only to be rejected later in the parse. This treatment of alternation differs crucially from the strategy of analysis in two-level morphology, where lexical search is based on single-symbol identity of the initial search paths of several items (letter trees, cf. e.g. Koskenniemi 1983, p. 107ff) and therefore "blind" to the individual orthographic properties of lexical items at search time.

2. THE STRUCTURE OF LEXICAL ITEMS

The formal declaration of individual lexical items is fairly simple: An item is declared as a line containing four elements: i. an *input string identifier*, ii. an *output string*, iii. a *left feature specification*, and iv. a *right feature specification*.

The excerpt (6) from the lexicon partition endw (containing endings) in the Danish TSS-system illustrates the declaration structure for lexical items.

(6)

```
i       ii          iii       iv

en~     +0n         NCA     / NCA /
en~     -0n         NCB     / NCB /
er~     +0r         PER     / PER /
et~     +0d         NNA     / NNA /
et~     -0d         NNB     / NNB /
e~      !0          AE      / AE  /
e~      -0          PE      / PE  /
ne~     n0          PER PE  / PD  /
ene~    +0n0        SER PNO / PD  /
s~      +s          N A P   / GEN /
t~      !t          AN      / AN  /
~       `           NCA     / NCA /
```

Element i, the input string identifier, is one of the graphemic alternants of the morpheme. For items which do not exhibit such alternation this string is simply the orthographic form of the morpheme; for alternating items the input string identifier is that alternant whose structure is described as @I0 in the alternation specification text of the lexicon partition to which the item belongs, cf. above. The items in (6) all end in the ~ (tilde). This is because they happen to be endings: the tilde matches "end-of-word", i.e. any sequence of blanks or an "end-of-input" signal. In a parsing system without any distinction between morphology and syntax such a character is necessary, since any character is taken to be a relevant part of the orthographic surface structure.

The input string identifier of a lexical item may be an empty string. In the Danish lexicon system a lexicon partition declared as bssr contains items occurring as "linking morphemes" between two parts of a compound. This lexicon partition only contains three items which are declared as in (7):

(7)

```
`       #       CD  / ! /
e       -0#     CE  / ! /
s       +s#     CS  / ! /
```

The first of these items has an empty string as its input string identifier. For reasons of readability an empty string is identified as the symbol `. The "morpheme" in question is used to take care of the fact that several Danish roots appear without any (non-empty) linking morpheme.

Formally it is a genuine lexical item, and its left feature specification, CD, is in fact responsible for the accept of a compound like *vandrør* 'water-pipe' and the rejection of an ill-formed compound like **buksvand*.

Element ii is the *output representation* of the item, i.e. that representation of the morpheme which is concatenated with the corresponding representations of neighbouring morphemes in the parsed structure. In the TTS-system for Danish the output representation of lexical items is *morphophonemic* in the linear sense of SPE-like phonological descriptions, (Chomsky & Halle 1968), i.e. it is a *sequence of phonetically interpretable symbols optionally surrounded by boundary symbols*. This output format is a sensible choice in such a system, due to the trivial fact that the phonetic representation of a single morpheme in a specific context can not be determined independently of that context, which is the very reason why a phonological component is needed. In principle, however, any output representation is the linguist's choice.

A comparison with the format of the lexical strings which are the output representations in two-level morphology is in order here. In two-level morphology the lexical representations contain certain arbitrary symbols ("features", see Koskenniemi 1983, p. 24) whose function is to form contexts for alternation rules which influence the accept or rejection of a given item in a given word form, i.e. the lexical representations are partly determined by factors relevant to the morphemic identification, hence to the result of the morphological analysis itself. In SSPS - where graphemic alternation is declared in the *alternation specification text* - there is no connection whatsoever between the analysis and the specific format of the output representation. The linguist is free to base the output representations on whatever considerations he sees fit, but in TTS-systems some sort of morphophonemic representation is the natural choice.

Elements iii and iv are the *feature specifications* of the item. In order for the system to treat features correctly, the features - like the lexicon partitions and their alternation patterns - must be declared in the declaration text. Features are declared by entering lines consisting of the keyword FEATURE followed by a feature name which must be a string of capital letters, e.g. thus:

FEATURE NNA

Each feature name declared in the declaration text refers to a *unary feature*, i.e. to a single-valued property; in other words, the SSPS feature system is not of the *attribute-value* type used in e.g. the D-PATR formalism (Karttunen 1986). It is possible, however, to refer to *groups of defined features*, because *a feature symbol in lexical items and in grammar rules refers to all defined unary features whose names begin with the symbol.* In other words, the convention for referring to lexicon partitions holds for feature references too: if four features are defined in the declaration file as

```
FEATURE NNA
FEATURE NCA
FEATURE NNB
FEATURE NCB
```

then the feature symbol N in a feature specification in the grammar or in the lexicon refers to all four features, NN refers to NNA and NNB, NC refers to NCA and NCB, NNB refers only to NNB, etc. A feature specification in the declaration of a lexical item is a sequence of blank-separated *feature names* delimited to the right by the character /. An exclamation mark - designating "presence of all features" - is also legal as a feature specification, as in (7). This may be used to express "free combinability" of sister constituents, cf. subsection II C.

The linguist may use features for whatever purposes he likes, but for parsing purposes features can be fruitfully used to combine *combinatorial* and *categorial* properties. The combinatorial viewpoint is primarily relevant for the morphological behaviour of items, whereas the categorial viewpoint is relevant to the syntactic properties of the items and of the higher-level constituents into which they enter as terminal constituents, cf. subsection II C and section III. The division of lexical feature specifications into a right part and a left part is primarily motivated by the combinatorial properties of morphemes within the word: this division reflects the fact that many morphemes have "janus properties" from the point of view of their combinability with other morphemes. This is most obvious in the case of suffixes: a suffix like *-ning* which forms noun stems from verbal roots is entered (in its appropriate lexicon partition) as

ning *niN+ V / NCA PER CSS /

The left feature specification is here simply V which specifies that this item is combinable with left sister constituents with verbal features (features whose name begin with V) in their right feature specification, cf. section II C. The right feature specification contains features specifying the nominal properties of the suffix, namely that it acts like a common gender noun (NCA) with plural *-er* (PER) and with obligatory *-s-* as a linking morpheme when it occurs as the first part of a compound (CSS), cf. *redningen - redninger - redningsbælte* 'salvation (sing. and plur.) - lifebelt'. This "directional" use of features is related to Whitelock's (1988) treatment of "signs".

Besides expressing combinatorial and categorial properties of lexical items, the feature specifications play an important role in connection with the grammar rules, as will be made clear in the next section.

C. The Grammar Formalism

The grammar formalism permits the linguist to write a constituent structure grammar with facilities for expressing *combinability restrictions* and *feature percolation* (cf. e.g. Lau & Perschke 1987), i.e. lexical feature specifications may be moved to mother nodes under conditions controlled by the grammar writer.

The skeleton of the grammar formalism is a context free grammar, i.e. a set of rules which rewrite nonterminal symbols on the left side of the rewrite symbol (in the examples the symbol ->) as a sequence of symbols specified on the right side of the rewrite symbol. The usual notational conventions for specifying optionality and repetition are legal: + after a right-side symbol means one or more occurrences of that symbol; ? means zero or one occurrence, and * (Kleene star) means zero or more occurrences. Likewise, the usual convention of designating terminal symbols by initial lowercase-letters and nonterminal symbols by initial uppercase letters is followed. As mentioned above, terminal symbols refer to lexical items from lexicon partitions whose names consist of or begin with the symbol.

In the following I presuppose familiarity with the basic formal properties of context free grammars, and I will confine myself to explaining those properties of the SSPS grammar formalism which are non-trivial. Examples are taken from the existing TTS-implementation for Danish.

1. SYLLABLE COUNT

After the left-side symbol of a rule there may follow a number. Such a number designates the minimal number of syllables (defined as orthographic vowels) required for the structure (subtree) represented by the left side symbol to be possible. From the point of view of Danish word structure a rule like (8) expresses the fact that stems composed of a prefix and a root always contain at least two syllables.

(8)

```
STEM 2  -> pn rn
```

From the point of view of parsing this facility represents an optimization: rule (8) tells the parser not to try to build this structure if the remaining part of the input text contains less than two syllables.

2. FEATURE PERCOLATION

Every lexical item in SSPS has two feature specifications, a left one and a right one, and *so has every constituent* in the tree structures described by the grammar.

Before I describe how constituents, i.e. nodes in the tree structures described by the grammar, acquire their feature specifications, I must explain an important convention for the interpretation of rewrite rules:

(9) *It is implicitly assumed that the structure described by a rewrite rule is legal if and only if it is true of any constituent (represented by any right-side symbol in the rule) that its left feature specification is compatible with the right feature specification of its left sister and that its right feature specification is compatible with the left feature specification of its right sister. For two feature specifications to be compatible they must share at least one unary feature, i.e. the set-theoretical intersection of the two feature specifications must not be empty.*

How do constituents acquire their feature specifications? *Terminal constituents* inherit their feature specifications from the lexical items with which they match, and I will therefore illustrate the meaning of this with rule (8) considered in connection with two strings of terminal material: *ufri* and *ugå*. Since *u* appears in the lexicon partition pn, and *fri* and *gå* appear in the lexicon partition rn, rule (8) would generate both these words (and the parser would accept them) if (9) were ignored. However, the right feature specification of *u* is A (standing for adjectival features, i.e. formally any feature whose name begins with A), and features of this kind (actually features named AC, AE, and AN) are also present in the left feature specification of *fri*, but not in the left feature specification of *gå*. As a consequence, since convention (9) is actually assumed, *ufri* is a legal structure, but *ugå* is not, and the parser would accept the string *ufri* as the corresponding word, but reject *ugå*.

Nonterminal constituents acquire their feature specifications in either of two ways: *If no explicit features are mentioned in a rule* (cf. below), a set of default conventions guarantees that any nonterminal constituent gets both a left and a right feature specification. These implicit conventions may be stated as follows:

(10) *Any mother constituent acquires the right feature specification of her rightmost daughter.*

(11) *Any mother constituent copies her left feature specification from her right feature specification.*

Principles (10) and (11) represent *implicit feature percolation*.

(10) expresses "rightheadedness" as a default principle (Selkirk 1982).

This principle guarantees, for example, that suffixed words like *redning* get the feature specification of their right member, in this case that the stem as such gets a right feature specification with the features NC etc., (cf. above) percolated from *-ning*.

3. EXPLICIT FEATURE MANIPULATION IN RULES

A basic grammar symbol is a string of letters, the first of which is upper-case if the symbol is nonterminal, otherwise lower-case. Before and after a basic grammar symbol a *modifier* may appear. A modifier is either a *percolator* or a *restriction*. A percolator is one of the symbols ^ >. A restriction has the following formal syntax:
a left parenthesis + an optional *restrictor sequence* + a right parenthesis.
A restrictor sequence consists of one or more *restrictors* separated by semicolons.
A restrictor consists of a *restrictor operator* optionally followed by a *restrictor operand.*
A restrictor operator is one of the symbols = # ? % : & + -.
A restrictor operand consists of one or more *feature symbols* separated by commas.
A feature symbol is a string of capital letters or an exclamation mark, i.e. its formal structure is that of lexical feature specifications.

A restrictor sequence which mentions features refers to the features of the left feature specification of the constituent in question if the restrictor sequence is written at the left side of the basic symbol, and to the right feature specification if it is written at the right side of the symbol. A basic grammar symbol with a right-sided restriction may, for instance, look like this:

STEM(:NN,PN)>

where the basic symbol is STEM which is modified by the right-side restriction (:NN,PN) and the percolator >.

The function of percolators and restrictions is to override the above-mentioned default conventions concerning the combinability of sister constituents and the feature percolations to mother constituents. Let me illustrate the most important functions of such explicit modifiers:

Explicit percolation may be *horizontal* (designated by the percolator symbol >) or *vertical* (designated by the percolator symbol ^). Explicit horizontal percolation copies the feature specification of a constituent to the corresponding feature specification of its right sister, carries out a logical AND-operation with the sister's feature specification, and leaves the result, i.e. the intersection of the two original feature specifications, as the sister's feature specification. A rule like

Word -> STEM> endw

declares for instance, that if STEM has inherited the right feature specification AAA BBB and endw has inherited the right feature specification BBB CCC, then, in the subtree described by the rule, endw must have the right feature specification BBB (due to the explicit horizontal percolation). Word, too, must have the BBB as both right and left feature specification, due to default feature percolation from the rightmost daughter (10) and to the copying convention (11).

Explicit vertical percolation is used to override the default "rightheadedness" principle. A rule like

NP -> ^N^ PP

makes N the head of NP in that both its left and right features (instead of the features of the rightmost daughter PP) are percolated to the mother NP. Note that this is the natural description of e.g. English noun phrases like 'the man with the red hat'. The entire noun phrase has the features of 'man', including e.g. features designating *3. person* and *singular* which are relevant for subject-verb agreement in English. Rightheadedness is predominant in morphology, it is not so frequent in syntax. The rule

NP -> ^N PP

overrides the principle that a mother copies her left feature specification from her right feature specification. In this case NP gets the left feature specification of N (due to explicit percolation) and the right feature specification of PP (due to implicit percolation).

The restrictors all have an operator and a feature operand. In the explanations given below of the functions of restrictors the following abbreviations will be used:
CON = the basic grammar symbol representing the constituent subject to the restriction.
OF = the original, i.e. inherited or percolated, feature contents of the relevant (left or right) feature specification of the constituent in question.
GF = the feature operand of the restrictor.
RF = the feature contents of the relevant feature specification resulting from the operation. Note that OF etc. have the formal syntax FFF (in the case of a single unary feature) or FFF,GGG,... (in the case of a combination of unary features) where FFF and GGG are feature symbols.

The operators =, #, ?, and % express *conditions for the acceptability* of the constituent in the subtree corresponding to the rule.

CON(=GF) means "CON is only legal if OF = GF"
CON(#GF) means "CON is only legal if OF =/= GF"

CON(?GF) means "CON is only legal if GF is included in OF"
CON(%GF) means "CON is only legal if GF is not included in OF"

The operators :, &, +, and -, express *explicit deviations from the default feature specifications* of the constituent in question.

CON(:GF) means "assign GF to RF"
CON(&GF) means "assign the intersection of OF and GF to RF"
CON(+GF) means "assign the union of OF and GF to RF"
CON(-GF) means "assign (OF minus GF) to RF"

If there are several (semicolon-separated) restrictors in a restrictor sequence, the operations may be thought of as being carried out in the order left to right. Thus CON(&FFF,GGG;-HHH) means "replace the original (inherited or percolated) contents of the right feature specification of CON with the intersection of those contents and FFF,GGG; then subtract HHH from the result and assign the new result to RF". Regarded as a declaration of the legality of a constituent in a subtree, such a restrictor series should be interpreted as the *final result*, i.e. the declaration says that the constituent is legal if the relevant feature specification has the contents which would be the result of this series of operations.

After this *tour de force* through the main formal properties of the lexicon and grammar formalism, we are in a position to study their use in the description of Danish surface structure.

III. SSPS AND DANISH SURFACE STRUCTURE

In this section I will illustrate the use of the SSPS formalism in declarations of morphological and surface syntactic structures in Danish. The rules and declarations may also be interpreted as instructions to the SSPS *parser*, cf. section IV.

I will illustrate various aspects of the SSPS formalism by presenting a sample SSPS grammar (12) which describes simple sentences as having a rather "flat" structure. Some of the constituent names refer to *fields* in Diderichsen's (1962) structural *field grammar* which is of the "slot and filler" type (Winograd 1983, p. 79). For ease of reference the rules of the grammar are numbered.

(12)

```
1  S     2  ->     NP(:!) (?VFA)WORD NP?(:!) PREP?
2  NP    2  ->     DETR?> (-N)DESC? KERN(:!) PREP?
3  PREP  2  ->     prep NP
4  DETR  1  ->     detr
5  DETR  2  ->     detr?> NUM(&A,PE)
6  DETR  1  ->     NP geni(:!)
7  NUM   1  ->     numri numr*
8  DESC  1  ->     (?A)WORD+
9  KERN  1  ->     (?N,P)WORD
10 WORD  1  ->     STEM endw
11 WORD  2  ->     STEM bssw(:!) (:!)STEM endw
12 WORD  3  ->     STEM bssw(:!) (:!)STEM bccw(:!) (:!)STEM endw
13 STEM  1  ->     rnr
14 STEM  1  ->     STEM snr
15 STEM  2  ->     pnr(?V) (?V)STEM
16 STEM  2  ->     pnr(?V) (%V)STEM(:VED,VET)
17 STEM  2  ->     pnr STEM(-V)
```

These 17 rules describe simple sentences, partly in field grammar terms, with an NP (the subject) in the "front field" (Diderichsen's *fundamentfelt*), with a finite verb as the only filler in the "verbal field" (Diderichsen's *nexusfelt*), and with an optional noun phrase (the direct object) followed by an optional prepositional phrase in the "content field" (Diderichsen's *indholdsfelt*).

The meanings of the non-trivial constituent names of the NP are the following:
DESC is a "descriptor field" (Diderichsen's *beskriverfelt*)
DETR is a "determiner field" (Diderichsen's *bestemmerfelt*)
KERN is a "kernel field" (Diderichsen's *kernefelt*)
The names of the nonterminal morphological constituents are self-evident, I hope. The terminal symbols refer to items from the lexicon partitions listed in (13):

(13)

```
prep     prepositions
detr     determiners (articles, quantifiers, etc.)
numr     numeral morphemes
numri    numeral morphemes occurring initially
geni     the genitive ending
endw     declensional endings
rnr      native root morphemes
bssw     linkers in simple compounds
bccw     linkers in "deep" compounds
snr      native suffixes
pnr      native prefixes
```

A remark on the use of features will help the reader to better understand some of the examples given in this section.

Formally, a declared feature name signifies nothing but the existence in the system of a certain unary feature, and it is the SSPS user's responsibility to use features consistently and meaningfully. A special hint for users of SSPS is, however, in order here: in many cases the same feature may be used with different interpretations in morphology and syntax, since these two levels - though formally indistinct in SSPS - are in most languages complementary as to the roles of features. There is nothing to prevent the user from using a feature XX as, say, a conjugation class marker in morphology and as, say, a marker of definiteness in syntax. Endings play a particular role in this respect in the SSPS description of Danish used for the TTS-parser: Since left and right feature specifications are distinct, endings may be assigned morphologically relevant left features and syntactically relevant right features.

The features mentioned in this section are listed in (14) with two interpretations, one for morphology (M) and one for syntax (S).

(14)

	M	S
VFA	past tense in -te	finite verb
PE	plural in e	indefinite noun, plural
PD	plural in er	definite noun, plural
AC	adjectival zero	common indefinite adj. sing.
AE	adjectival e	definite or plural adj.
AN	adjectival t	neutral indefinite adj. plur.
NNA	neutral noun in zero	neutral indefinite noun, sing.
NNB	neutral noun in e	neutral definite noun, sing.
NCA	common noun in zero	common indefinite noun, sing.
NCB	common noun in e	common definite noun, sing.

In the grammar (12) rules 1 - 9 describe the syntactic part of such structures. Rules 10 - 17 describe the "morphological" part. I do not intend to explain every detail in (12), but I will comment on a handful of characteristic properties of a some of these rules.

The restrictor (:!) after the initial NP in rule 1 declares that a noun phrase combines freely with a finite verb. This is the SSPS way of stating the fact that there are no agreement-like dependences between subject and verb in Danish.

The *finite verb* is represented by the symbol (?VFA)WORD in rule 1, i.e. the word class property of the category WORD appears as a feature (VFA meaning "finite") which is percolated from the internal constituents of the category, ultimately from lexical items. Likewise, note the identification of a *noun* as a (N,P)WORD, i.e. a word with the (left) feature symbols N or P in rule 9. These symbols "unify" nominal features referring to singular and plural declensional classes which are relevant in the morphological part of the grammar, but this "unification" is accomplished simply by the "abstract" use of feature symbols made possible by the naming conventions mentioned in section II. In this case all unary features whose names begin with N or P are covered, but the only thing that matters from a syntactic point of view is to identify a noun as such, so the full "morphological" specification is simply left out here; cf. also the identification of *one or more adjectives* as (?A)WORD+ in rule 8.

Another illustrating aspect of this grammar is the treatment of the dependency between the constituents DETR, DESC, and KERN in the NP of rule 2. A Danish noun phrase is either definite or indefinite. The definiteness is expressed in either of two ways, depending on the structure of the NP: if the noun phrase consists of an isolated noun, the definite form of that noun (*manden* 'the man' vs. *mand* 'man') is responsible for the

definiteness. If, however, the NP is modified by a determiner followed by an adjective, the definiteness or indefiniteness is expressed solely by that determiner, and in this case the noun is always in the indefinite form, whereas the form of the adjective depends on the determiner. If the determiner is *indefinite*, the adjective must agree in number and gender with the *noun*: *en god mand* 'a good man'; *et godt skib* 'a good ship'; *nogle gode skibe* 'some good ships', and this is also the case if there is no determiner at all: *godt vejr* 'good weather', *god kaffe* 'good coffee', *gode skibe* 'good ships'. If the determiner is definite, however, the adjective must agree in *definiteness* with the *determiner*: *den gode mand* 'the good man'; *det gode skib* 'the good ship'.

I will show in some detail how the choice of features in the lexicon and the manipulation of features in the grammar may be combined to take care of these phenomena.

Consider the following fragments from lexicon partitions (LP's) in (15).

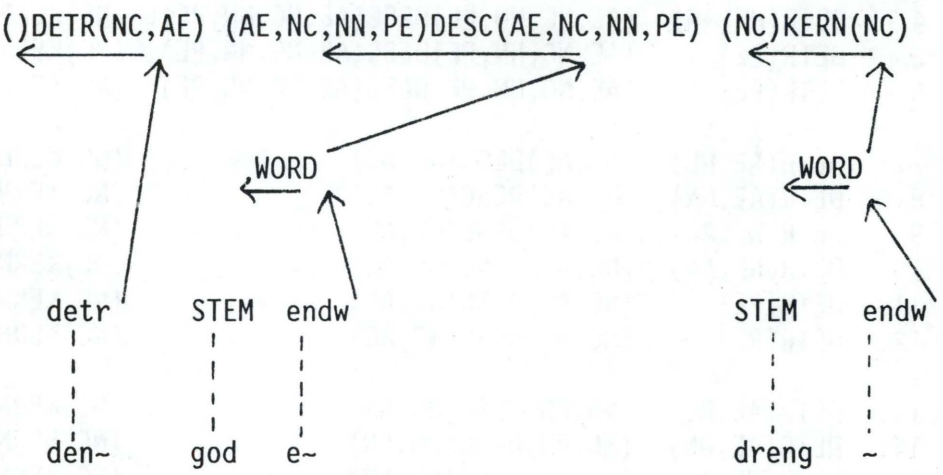
(15)

```
LP: rnrr (* non-alternating roots *)
god       go:d        ! / AC AN AE /
dreng     dræN        !  / NC PE CE /


LP: detr (* non-alternating, unstressed determiners *)
den~      dænh%       !  / AE /
det~      de%         !  / AE /
en~       enh%        !  / NC AC /
et~       eth%        !  / NN AN /
de~       di%         !  / PE /
nogle~    nol0%       !  / PE /


LP: endw  ( *endings* )
~         '           NC / NCA /
~         '           NN / NNA /
~         '           AC / AC /
e~        -0          AE / AE NC NN PE /
e~        -0          PE / PE /
t~        +t          AN / AN /
ne~       n0          P  / PD
```

Consider next the NP 1. *den gode dreng* 'the good boy': due to principles (10) and (11) of section II, and due to the fact that no rules below the NP-level in (12) override these principles for the structure in question,

the lexical feature specifications of the terminal constituents of this NP are percolated through the "middle" constituents (STEM and WORD) to the higher constituents DETR, DESC, and KERN, as illustrated in (16) where - for reasons of space - the irrelevant feature specifications at the top level and the feature specifications of the terminal (lexical) and middle constituents are omitted.

(16)



(16) shows what the structure *just below the NP-level* would have looked like if the right-percolator (>) to the right of DETR and the "subtractor" restrictor (-N) to the left of DESC in rule 2 had not been there, that is if rule 2 had looked like

2 NP 2 -> DETR? DESC? KERN(:!) PREP?

All the lower level constituents simply percolate their right feature specifications to their mothers according to (10), and the mothers copy their right feature specifications to their left ones according to (11), as indicated by the arrows.

Consider now the following NP's of which most are illegal:

2. **det gode dreng* 3. **en gode dreng* 4. **et gode dreng* 5. **de gode dreng* 6. **nogle gode dreng* 7. **den god dreng* 8. **det god dreng* 9. *en god dreng* 'a good boy' 10. **et god dreng* 11. **de god dreng* 12. **nogle god dreng* 13. **den godt dreng* 14. **det godt dreng* 15. **en godt dreng* 16. **et godt dreng* 17. **de godt dreng* 18. **nogle godt dreng* 19. **den gode drenge* 20. **det gode drenge* 21. **en gode drenge* 22. **et gode drenge* 23. *de gode drenge* 'the good boys' 24. *nogle gode drenge* 'some good boys' 25. **den gode drengene* 26. **det gode drengene* 27. **en gode drengene* 28. **et gode drengene* 29. **de gode drengene* 30. **nogle gode drengene*

On the assumption, still, that rule 2 has been changed in the indicated way, the situation at the level in question for these structures may be schematized as in (17):

(17)

1.	DETR(AE,NC)	(AE,NC,NN,PE)DESC(AE,NC,NN,PE)	(NC)KERN
2.	DETR(AE,NN)	(AE,NC,NN,PE)DESC(AE,NC,NN,PE)	(NC)KERN
3.	DETR(NC,AC)	(AE,NC,NN,PE)DESC(AE,NC,NN,PE)	(NC)KERN
4.	DETR(NN,AN)	(AE,NC,NN,PE)DESC(AE,NC,NN,PE)	(NC)KERN
5.	DETR(PE)	(AE,NC,NN,PE)DESC(AE,NC,NN,PE)	(NC)KERN
6.	DETR(PE)	(AE,NC,NN,PE)DESC(AE,NC,NN,PE)	(NC)KERN
7.	DETR(AE,NC)	(NC,AC)DESC(NC,AC)	(NC)KERN
8.	DETR(AE,NN)	(NC,AC)DESC(NC,AC)	(NC)KERN
9 .	DETR(NC,AC)	(NC,AC)DESC(NC,AC)	(NC)KERN
10.	DETR(NN,AN)	(NC,AC)DESC(NC,AC)	(NC)KERN
11.	DETR(PE)	(NC,AC)DESC(NC,AC)	(NC)KERN
12.	DETR(PE)	(NC,AC)DESC(NC,AC)	(NC)KERN
13.	DETR(AE,NC)	(NN,AN)DESC(NN,AN)	(NC)KERN
14.	DETR(AE,NN)	(NN,AN)DESC(NN,AN)	(NC)KERN
15.	DETR(NC,AC)	(NN,AN)DESC(NN,AN)	(NC)KERN
16.	DETR(NN,AN)	(NN,AN)DESC(NN,AN)	(NC)KERN
17.	DETR(PE)	(NN,AN)DESC(NN,AN)	(NC)KERN
18.	DETR(PE)	(NN,AN)DESC(NN,AN)	(NC)KERN
19.	DETR(AE,NC)	(AE,NC,NN,PE)DESC(AE,NC,NN,PE)	(PE)KERN
20.	DETR(AE,NN)	(AE,NC,NN,PE)DESC(AE,NC,NN,PE)	(PE)KERN
21.	DETR(NC,AC)	(AE,NC,NN,PE)DESC(AE,NC,NN,PE)	(PE)KERN
22.	DETR(NN,AN)	(AE,NC,NN,PE)DESC(AE,NC,NN,PE)	(PE)KERN
23.	DETR(PE)	(AE,NC,NN,PE)DESC(AE,NC,NN,PE)	(PE)KERN
24.	DETR(PE)	(AE,NC,NN,PE)DESC(AE,NC,NN,PE)	(PE)KERN
25.	DETR(AE,NC)	(AE,NC,NN,PE)DESC(AE,NC,NN,PE)	(PD)KERN
26.	DETR(AE,NN)	(AE,NC,NN,PE)DESC(AE,NC,NN,PE)	(PD)KERN
27.	DETR(NC,AC)	(AE,NC,NN,PE)DESC(AE,NC,NN,PE)	(PD)KERN
28.	DETR(NN,AN)	(AE,NC,NN,PE)DESC(AE,NC,NN,PE)	(PD)KERN
29.	DETR(PE)	(AE,NC,NN,PE)DESC(AE,NC,NN,PE)	(PD)KERN
30.	DETR(PE)	(AE,NC,NN,PE)DESC(AE,NC,NN,PE)	(PD)KERN

Of the illegal NP's 13 - 18 and 25 - 30 would be rejected as they should: 13 - 18 would be rejected because the right feature specification (NN) of DESC is not compatible with the left feature specification (NC) of KERN (cf. principle (9)), and 25 - 30 would be rejected for similar reasons. But there would still be considerable overgeneration: the illegal NP's 2-8, 10-12, and 19-22 would be accepted, because any two contiguous right and

left feature specifications are compatible (share at least one unary feature).

Consider now the effect of (and the motivation for) the restrictions of the "real" rule 2, namely the right-side horizontal percolation (>) of DETR and the left-side "subtractor" restrictor (-N) of DESC, cf. (18), where the illegal structures are marked with *.

(18)

1.	DETR(AE,NC)	(AE,PE)DESC(AE,NC)	(NC)KERN
2.	DETR(AE,NN)	(AE,PE)DESC(AE,NN)	(NC)KERN *
3.	DETR(NC,AC)	(AE,PE)DESC(NC)	(NC)KERN *
4.	DETR(NN,AN)	(AE,PE)DESC(NN)	(NC)KERN *
5.	DETR(PE)	(AE,PE)DESC(PE)	(NC)KERN *
6.	DETR(PE)	(AE,PE)DESC(PE)	(NC)KERN *
7.	DETR(AE,NC)	(AC)DESC(NC)	(NC)KERN *
8.	DETR(AE,NN)	(AC)DESC()	(NC)KERN *
9 .	DETR(NC,AC)	(AC)DESC(NC,AC)	(NC)KERN
10.	DETR(NN,AN)	(AC)DESC()	(NC)KERN *
11.	DETR(PE)	(AC)DESC()	(NC)KERN *
12.	DETR(PE)	(AC)DESC()	(NC)KERN *
13.	DETR(AE,NC)	(AN)DESC()	(NC)KERN *
14.	DETR(AE,NN)	(AN)DESC(NN)	(NC)KERN *
15.	DETR(NC,AC)	(AN)DESC()	(NC)KERN *
16.	DETR(NN,AN)	(AN)DESC(NN,AN)	(NC)KERN *
17.	DETR(PE)	(AN)DESC()	(NC)KERN *
18.	DETR(PE)	(AN)DESC()	(NC)KERN *
19.	DETR(AE,NC)	(AE,PE)DESC(AE,NC)	(PE)KERN *
20.	DETR(AE,NN)	(AE,PE)DESC(AE,NN)	(PE)KERN *
21.	DETR(NC,AC)	(AE,PE)DESC(NC)	(PE)KERN *
22.	DETR(NN,AN)	(AE,PE)DESC(NN)	(PE)KERN *
23.	DETR(PE)	(AE,PE)DESC(PE)	(PE)KERN
24.	DETR(PE)	(AE,PE)DESC(PE)	(PE)KERN
25.	DETR(AE,NC)	(AE,PE)DESC(AE,NC)	(PD)KERN *
26.	DETR(AE,NN)	(AE,PE)DESC(AE,NN)	(PD)KERN *
27.	DETR(NC,AC)	(AE,PE)DESC(NC)	(PD)KERN *
28.	DETR(NN,AN)	(AE,PE)DESC(NN)	(PD)KERN *
29.	DETR(PE)	(AE,PE)DESC(PE)	(PD)KERN *
30.	DETR(PE)	(AE,PE)DESC(PE)	(PD)KERN *

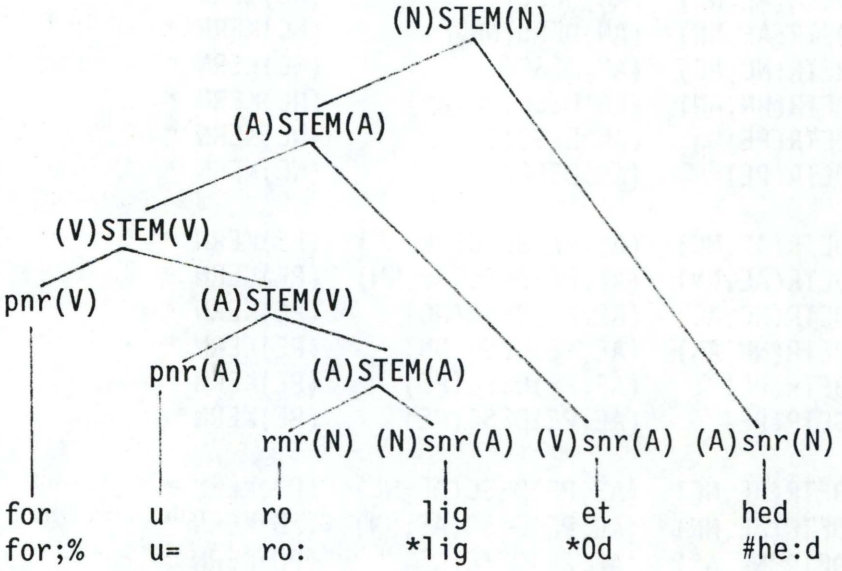
Thanks to the restrictions of rule 2, all the legal structures are accepted,

and all the illegal ones are rejected.

I am aware that this may be hard to see from the grammar (mainly because of the implicitness of feature percolation), but I only use this example to demonstrate the ability of SSPS to express rather complicated dependencies in a compact way. Incidentally, this property is relevant to the speed of the parser, which depends more on the number of rules to try than on the conceptual complexity of the rules. Note that the percolator > and the restrictor (-N) in rule 2 are not just ad hoc formal devices: the natural linguistic interpretation of the horizontal percolator > may be formulated thus: "If the determiner field and the describer field are both present, they *combine* to form the definiteness value of a Danish noun phrase", and the natural linguistic interpretation of the restrictor (-N) may be formulated thus: "If the determiner field and the describer field are both present, nominal agreement features of the describer field are *ignored* in a Danish noun phrase".

In the *morphological* part of the grammar (12) attention should be paid to rules 15-16. These rules are recursive and describe the structure of such "deep" morphological structures as (19), where both the input and output formats of the terminal constituents are shown, and where the most relevant (abstract) left and right features are shown in parentheses.

(19)



Notice the restrictors in rules 15 and 16. In normal prose, what rule 15 says is: a STEM may consist of a native prefix with verbal right features followed by a native root; if the root has verbal left features, normal rightmost daughter percolation takes place, i.e. the STEM will be a verbal

stem like *be-søg* 'visit'. This guarantees that STEM will have the conjugation class of the root *søg*, in particular it will be marked for the past ending *-te* (the feature VFA in its morphological interpretation, cf. (14) above).

Rule 16 says: a STEM may consist of a native prefix with verbal right features followed by a native root; if the root has no verbal left features, its right feature specification will be the combination (VED, VET) which are percolated implicitly to the mother STEM. This rule caters for the fact that many nominal and adjectival roots (and stems) may be "verbalized" by verbal prefixes, and that such verbs have the *unmarked conjugation* (past tense *-ede* and past participle *-et*), as expressed by the features VED and VET, cf. e.g. *afkviste* 'to cut off twigs', literally "to off-twig".

These comments have, I hope, served as good illustrations of the expressive facilities of SSPS, and of the linguistic meaningfulness (interpretability) of restrictors and percolators.

IV. THE SSPS PARSER IN OUTLINE

The parser used in the Danish TTS-system is tuned to the SSPS formalism. I will limit myself to outlining its main general features. The parser is based on the *active chart* principle (Earley 1970; Winograd 1983, p. 116ff is a good introduction), and proceeds in a *top-down, depth first, first rule first, first solution only, left to right* fashion.

The top-down principle was chosen on empirical grounds: a bottom-up version exists and has been used, but tests showed that the overgeneration of hypotheses at the lower level characteristic of bottom-up parsing exceeded the overgeneration near the top of the top-down version considerably. This undoubtedly has to do with the inclusion of morphology, which means that the terminal constituents are not given in advance, but must be identified during parsing. For the same reason, optimizations à la Wirén (1987) are not possible.

The depth first and first rule first principles were chosen because they are easy to combine with the principle of selecting the first solution found, and because they enable the user to order his grammar rules according to e.g. his knowledge of the frequency or probability of certain structures. This is possible because the parser simply processes the subtrees in the order of the corresponding rules in the grammar. Most Computational linguists today would contend that the grammar writer should be allowed to write his grammar without considerations of how a parser would handle the grammar in connection with input (the principle of purely *declarative systems*). I agree in the sense that the grammar writer should not be *forced* to consider how a parser or any other program "understanding" the formalism will treat a specific input. But SSPS *gives the grammar writer the*

option to exploit the first rule first principle in that he *may* order his rewrite rules in such a way as to arrive at a preferred structural interpretation first, which is quite different from being *forced* to consider parsing schedule. This possibility is important in a practical TTS- system, because only one solution should be handed down to the phonological and phonetic components and further down to the synthesis component. The first rule first principle is also well chosen in connection with unidentified input: The TTS-system *must* "say" something, and this requirement may be met by putting very "permissive" root symbol rules at the *bottom of the grammar*, so that they are tried after all "structured" rules, cf. a rule like

S -> (:!)WORD(:!)*

which simply says: "let any sequence of words be accepted". This is the SSPS way of arriving at preferred structural interpretations in cases of ambiguous input without necessarily rejecting improbable or downright illegal structural interpretations in cases of ill-formed input. To take an example: why should not a TTS-system for Danish assign the "pronunciation" [dɛn'go:ðə'sgi:b'sɑjlʌ]? to an improper input sentence like **den gode skib sejler*? Most Danish speakers would read it aloud that way.

The left-right strategy may not be the best one, cf. that "island parsing" seems to give good results in other fields of recognition of structure, especially speech recognition.

For the benefit of readers familiar with chart parsing, I may add that the evaluation of restrictions takes place in connection with the "subsumption" of complete edges by active ones: active edges about to "clone" themselves check the restrictions and act according to the results, which often is that the cloning is cancelled.

The parser performs fast enough to be functional in the TTS-system, where the bottleneck as far as execution time is concerned is still the synthesis component.

The inclusion of syntactic rules has meant a considerable reduction of misinterpretations of input which is ambiguous from a *word-level* point of view: in Danish heterophonic homographs (*hul, bad, så, dør, bred*, etc.) typically belong to different word classes (and thus have different feature specifications, cf. section III), and can therefore in many cases be disambiguated by a moderate surface syntactic analysis. With the grammar (12) and the present morpheme lexicon which comprises about 9000 - judiciously featured - items, the parser finds the correct interpretation of e.g. the input sentence "en mand med en *hul* røst bag en *bred dør* med et *hul dør*" 'a man with a hollow voice behind a wide door with a hole (in it) dies'.

V. PHONOLOGY IN TTS-SYSTEMS

The transformation of the linearized morphophonemic parser output strings to a phonetic transcription is described in another formalism, namely a trimmed and otherwise adapted phonological version of the SPL-language described by Holtse (1982) and closely related to the older SPE-like formalism of Carlson & Granström (1975). I will not describe the formalism here, since its properties are in a sense trivial, especially to readers familiar with TTS-methodology.

Rather, I would like to stress the fact that the extremely linear conception of phonology implicit in SPE-based formalisms is becoming obsolete in view of recent phonological theories, and, more importantly, in view of the hierarchical structure of both morphology and syntax. The SSPS framework permits the user to express hierarchical structuring of surface syntax and morphology, but the projection of such information on a line (in the form of more or less fancy (strings of) boundary symbols, cf. the examples of output formats in previous sections) is not particularly elegant, and it entails a good deal of clumsiness in formulations of e.g. phenomena like syntactically and semantically conditioned *unit accentuation* in Danish (see Rischel 1982).

One of the most important tasks for present-day speech technology is to design phonological (and phonetic) formalisms permitting the user to express the relations between syntactic surface structure and prosody - in particular stress patterns - in appropriate ways.

REFERENCES

Allen, J., Hunnicutt, S. M., and Klatt D. 1987: *From text to speech: the MITalk system*, Cambridge University Press.

Carlson, R. and Granström, B. 1975: "A text-to-speech system based on a phonetically oriented programming language", *Speech Transm. Lab., Quart. Prog. and Status Rep. 1/1975,* p. 17-26.

Chomsky, N. 1970: "Remarks on Nominalization", in Noam Chomsky: *Studies on Semantics in Generative Grammar*, Mouton.

Chomsky, N. and Halle, M. 1968: *The Sound Pattern of English*, Harper and Row.

Diderichsen, P. 1962: *Elementær Dansk Grammatik*, Gyldendal.

Earley, J. 1970: "An Efficient Context-free Parsing Algorithm," *Communications of the Association for Computing Machinery, 13(2),* p. 94-102.

Holtse, P. 1982: "Speech Synthesis at the Institute of Phonetics" *Ann. Rep. Inst. Phon. Univ. Cph. 16*, p. 117-126.

Karttunen, L. 1986: "D-PATR: A Development Environment for Unification-based Grammars", *Proceedings of the 11th International Conference of Computational Linguistics*, p. 74-80. International Commitee on Computational Linguistics.

Koskenniemi, K. 1983: *TWO-LEVEL MORPHOLOGY: A General Computational Model for Word-Form Recognition and Production*, University of Helsinki Publications, No. 11.

Lau, P. and Perschke, S. 1987: "Morphology in the Eurotra Base Level Concept," *Proceedings of the Third Conference of the European Chapter of the Association for Computational Linguistics*, p. 19-25. The Association for Computational Linguistics.

Molbæk Hansen, P. 1983: "An Orthography Normalizing Program for Danish", *Ann. Rep. Inst. Phon. Univ. Cph. 17*, p. 87-110.

Molbæk Hansen, P. (Forthcoming): "Nogle svagheder ved toniveaumorfologien", will appear in *Skrifter om Anvendt og Matematisk Lingvistik 14.*

Rischel, J. 1982: "On unit accentuation in Danish - and the distinction between deep and surface phonology", *Ann. Rep. Inst. Phon. Univ. Cph. 16*, p. 191-240.

Selkirk, E. O. 1982: *The Syntax of Words*, MIT Press.

Whitelock, P. J. 1988: "A Feature-based Categorial Morpho-Syntax for Japanese", in U. Reyle and C. Rohrer (eds.): *Natural Language Parsing and Linguistic Theories"*, p. 230-261.

Winograd, T. 1983: *Language as a Cognitive Process, Volume 1: Syntax*, Addison-Wesley Publishing Company.

Wirén, M. 1987: "A Comparison of Rule-Invocation Strategies in Context-Free Chart Parsing", *Proceedings of the Third Conference of the European Chapter of the Association for Computational Linguistics*, p. 226-233. The Association for Computational Linguistics.