Integrating Stress and Intonation into a Concept-to-Speech System

Georg DORFFNER
Ernst BUCHBERGER
Austrian Research Institute
for Artificial Intelligence
Schottengasse 3
A-1010 Vienna, Austria
and University of Vienna
email: georg%ai-vie.uucp@relay.eu.net,
ernst%ai-vie.uucp@relay.eu.net

Markus KOMMENDA
Institut f. Nachrichtentechnik
und Hochfrequenztechnik
Technical University of Vienna
Gusshausstr. 25/389
A-1040 Vienna, Austria
email: E389011@AWITUW01.BITNET

Abstract: The paper deals with the integration of intonation algorithms into a concept-to-speech system for German [1]. The algorithm for computing the stress hierarchy of a sentence introduced by Kiparski (1973) and the theory of syntactic grouping for intonation patterns developed by Bierwisch (1973) have been studied extensively, but they have never been implemented in a concept-to-speech system like the one presented here. We describe the back end of this concept-to-speech system: The surface generator transfers a hierarchical dependency structure of a sentence into a phoneme string by traversing it in a recursive-descent manner. Surface structures unfold while generation proceeds, which means that at no point of the process does the full syntactic tree structure exist. As they depend on syntactic features, both the indices introduced by the Kiparski (degrees of stress) and the Bierwisch (indexed border markers) formalism have to be inserted by the generator. This implies some changes to the original algorithms, which are demonstrated in this paper. The generator has been tested in the domain of an expert system that helps to debug electronic circuits. The synthesized utterances of the test domain show significant improvements over monotonous forms of speech produced by systems not making use of intonation information.

1. Introduction

The goal of the system, a part of which is described in this paper, was to synthesize speech utterances starting from a conceptual representation of the knowledge to be uttered (*concept-to-speech* system). Compared to speech reproduction, our approach is far more flexible. In contrast to text-to-speech synthesis (Frenkenberger et.al. 1988) on the other hand, our approach allows for an easier integration of prosodic elements, as syntactic data such as phrases and tree dependencies are directly available.

Appropriate formalisms for obtaining a basis for stress and pitch information were introduced by Kiparski (1973), who proposed an algorithm for computing a stress hierarchy for a whole sentence, and Bierwisch (1973), who showed how to determine pitch variation patterns depending on the phrasal structure of a sentence. Like Kiparsky's stress markers, the boundary indices introduced by Bierwisch can be computed from the syntactic structure of the sentence.

In this respect, concept-to-speech contrasts with text-to-speech systems: In text-to-speech synthesis – at least for the German language – it is virtually impossible to carry out a complete syntactic analysis because of the large number of ambiguities which can only be resolved at the semantic level. Thus, the derivation of prosodic information in existing text-to-speech systems is based on a very rudimentary syntactic analysis which consists in a purely linear segmentation of the input sentences (e.g. Kulas & Rühl 1982, Zingle 1982, Schnabel 1988, Frenkenberger et al. 1988).

In concept-to-speech synthesis, on the other hand, we are in a position to exploit the inherently available syntactic structure of the given text, so that we can apply the formalisms described by Bierwisch and Kiparsky.

Both processes are only theoretically developed and have not been fully implemented in a working system before. We have integrated these processes into the surface generator of our concept-to-speech system and applied some necessary changes and adaptations to them.

In this paper we concentrate on the computation of stress and intonation markers, integrated into the surface generation component. The reader interested in the overall structure of the system, an application domain and the first phase of generation which starts with concepts and produces

1) This work was supported by the Jubiläumsfonds der Oesterreichischen Nationalbank, as part of project no. 2901.

the input structure to the surface generator (henceforth 'deep structure') is referred to Dorffner, Trost & Buchberger (1988).

2. The Surface Generator

The deep structure which forms the input to the surface generator consists of a hierarchical structure of essentially two building blocks: CLAUSEs, which roughly correspond to entire sentences and PHRASEs like NPs, PPs or APs (fig. 1). A PHRASE can be modified by other

PHRASE

type	noun, adj
head	*lxm*wid, ...
mods	<phrase>, <clause>
feats	

PHRASE-FEATURES

det	def, indef, ...
betont	t, nil
vorfeld	t, nil
pron	t, nil
case	e-zero, ...
num	sing, plur, 10, ...

Fig. 1

PHRASEs or CLAUSEs, thus forming a hierarchical structure for complex utterances (Dorffner, Kommenda & Frenkenberger 1988).

Surface generation now works on this hierarchical structure of building blocks and transfers it into a surface structure consisting of phonemic strings which are subsequently synthesized. Our generator differs from the often encountered two-step approach - generate the syntactic tree with lexical items as its leaves and morphological and other features attached to them, then scan all its leaves and synthesize the lexical elements (see e.g. McDonald 1983) - in an important way, for reasons of efficiency and plausibility. The deep structure, as introduced above, was designed so as to already correspond to the surface structure of the sentence [1], except for aspects of order and function words. In other words, the (unordered) hierarchy of deep structure building blocks is isomorphic (after order has been imposed) to the syntactic tree structure of the surface sentence. This can be easily achieved in German, where constituent order is much less strict than in other languages, such as English. As a result of this property of German, the position of phrases within a sentence is not tied to their functional role and thus does not have to be reflected in the deep syntactic structure. This design of a deep structure as being isomorphic to surface structure implies a simplification in the surface generator, compared to the two-step approach mentioned above: The surface tree does not have to be produced entirely before lexical items can be synthesized, but can unfold while the hierarchy of building blocks is scanned recursively.

The process of surface generation is as follows: For each CLAUSE or PHRASE, a corresponding surface building block (e.g. an NP) is generated, depending on their features and lexical heads (fig. 2). Such a building block contains slots for either

NP-PP

prep	*det*	*mod1*	*head*	*mod2*
Lexem	*<detp>*	*<phrase>* *<clause>* *	*Lexem*	*<phrase>* *<clause>* *

Fig. 2

pointers to other building blocks or lexical items in their *correct order*. Now each slot can be scanned and synthesized (if it contains a lexical item) or recursively treated like the other building blocks (Fig.3, Dorffner Kommenda & Frenkenberger 1988).

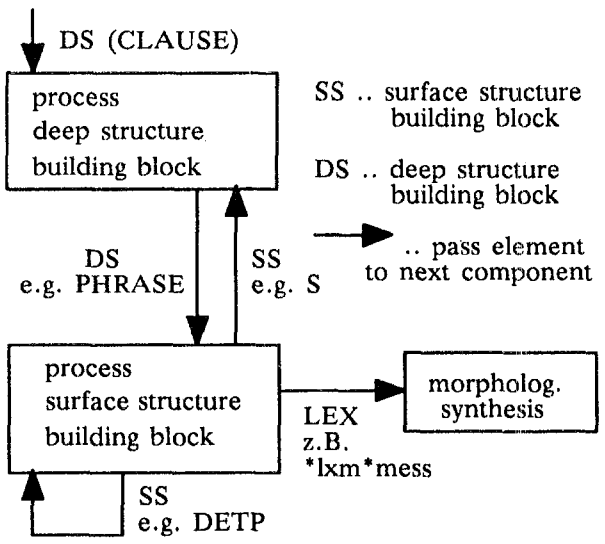


Fig. 3

This form of generation process has serious consequences on the intended integration of intonational information: All syntactic information

1) Strictly speaking, this differs from a deep structure as defined in Chomsky (1975)

is available during the process, but the syntactic tree never exists in its entirety. Furthermore, indices have to be produced (during synthesis of lexical items) *before* the remainder of the syntactic structure has unfolded. At first sight this looks like a major restriction and reduction of available information. As it turns out, however, the approaches of Kiparski and Bierwisch can both be modified so as to fit into this scheme. An interesting side-effect is that synthesis of speech, starting from deep structures, works in a strict left-to-right manner, which seems psychologically very plausible.

3. Insertion of Kiparski Stress Markers

Kiparski (1973) introduced two rules for computing stress markers based on a syntactic tree:

(1) (a) Head stress rule:
the first (left-most) node keeps its index, all others are incremented by 1
(b) Tail stress rule:
the last (right-most) node keeps its index, all others are incremented by 1

The algorithm works as follows:

(2) - assign the index 1 to each stressable lexical item
- scan the tree bottom-up and apply rule (1a) or (1b) to each significant node

This algorithm works strictly bottom-up and thus requires the entire syntactic tree. As a result, it cannot be integrated into our generator in this form. It is, however, possible to rewrite the algorithm so that it works top-down and depth-first so as to fit into the generation scheme described above. The new algorithm is the following:

(3) Introduce a pair of indices and maintain it as follows while scanning the tree top down. At the root, start with the pair (1 1).
- *at each significant node that has at least two significant successor nodes, do the following, given the index pair (n m):*
  - *with head stress rule:*
    assign the pair (n m+1) to the first successor
    assign the pair (n+m 1) to all the others
  - *with tail stress rule:*
    assign the pair (n m+1) to the last successor
    assign the pair (n+m 1) to all the others
- *at the leaves of the tree (= lexical entry), with assigned pair (n m):*
  - *n is the Kiparski marker for the lexical item*

If one considers the preferred successor (head or tail, depending on the rule) as the *winner* of the rule and all others as *losers*, algorithm (3) can be interpreted as follows: The second index of a pair (m) counts how often a node is on the winning side. All losers have to increment their marker by that amount. Thus, at each decision, the winner keeps its marker (n), while the markers of all the others have to be increased by m ($n+m$). As there can be only one leaf that is on the winning side each time, it is ensured that only one lexical item receives marker 1.

A similar algorithm could be applied to yield the stress pattern within complex words (which are quite numerous in German). However, as the lexicon of the generator contains morphemes and complex lexemes with pointers to each morpheme, a decision about stress within a word can be stored lexically and no algorithmic treatment is necessary. A syllable now receives a Kiparski marker if

- *it is in a stressable morpheme (lexical feature)*
- *it is marked by the lexical entry of the (possibly complex) word AND*
- *algorithm (3) has assigned an index pair to the lexical entry*

The so computed marker is inserted into the phonemic string during the morphologic synthesis of the word.

4. Insertion of Bierwisch Boundary Indices

Bierwisch (1973) suggests inserting a marker at each word boundary to express how many significant nodes dominate both words involved. His algorithm was designed in a bottom-up fashion. We show again that it can be formulated top-down (as required in our system):

(4) Assign an index to each node. At the root, start with 1. For each node with index i
for each successor do, left to right:
- *if the successor is a lexical item, synthesize it and append i as boundary marker*
- *if the successor is a significant node, assign index i+1*
- *otherwise assign index i*

when all nodes on that level have been processed,
- *overwrite the index that was written last with i*

The problem that a left-to-right process cannot know whether the following word is on the same level in the tree is solved by permitting to overwrite a marker already written.

5. Acoustic Realization of Prosodic Patterns

Starting from the above stress and boundary markers, the prosodic structure of a sentence is derived by applying a phonological rule set. In

particular, some of the previously computed boundaries are deleted, others receive a pause marker. Furthermore, the resulting phrases are provided with an intonation contour, which, according to Bierwisch (1973), is specified in terms of so-called SON values. In a subsequent phonetic component the phrasal structure and the SON values are exploited to generate the acoustic correlates of the prosodic information, in particular, the duration of phonetic segments and pauses and the pitch values for all voiced phones.

6. An Example

An annotated example shall illustrate the process of generation. Take the following sentence:

Beträgt die Spannung am Kondensator 10 Volt?
(Is the voltage at the capacitor equal to 10 Volts?)

The deep structure of the sentence, which is the input to the surface generator is depicted in fig.4,

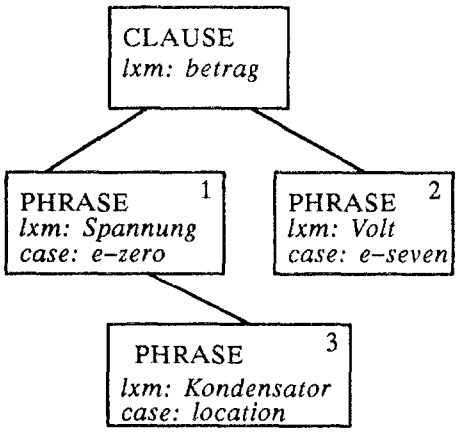


Fig.4

the corresponding syntactic tree, which is unfolded during generation, in fig.4a. Both structures have been simplified.

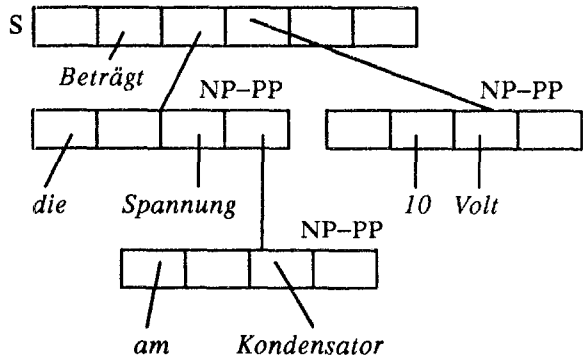


Fig.4a

Each building block in the dependency structure (to the left) has a feature *case* which indicates the conceptual role of the element (adapted from Engel 1982). *e-zero*, for example, refers to the nominative phrase or subject of a sentence. The structure to the right consists of the surface building blocks. Each slot (drawn as a box) corresponds to a possible position which can be filled with a lexical item or another building block, depending on the features of CLAUSE and PHRASE. Slots that remain empty are ignored during synthesis. One can see in this example that the tree of CLAUSEs and PHRASEs has a corresponding isomorphic tree of S and NP–PPs (there are other surface elements like AP, as well), with the exception that in the former case there is no order information yet. This illustrates the above mentioned isomorphism between deep and surface structure.

Generation starts at the root of the deep structure, the CLAUSE. A Kiparski pair (1 1) and a Bierwisch index 1 are assigned. The corresponding surface building block, S, is generated, filled with the lexical item *beträgt* (verb) and with the two PHRASES in their correct position (which can be determined by looking at the features and using some default heuristics as in Engel 1982). The structure at this point looks like the one in fig.5:

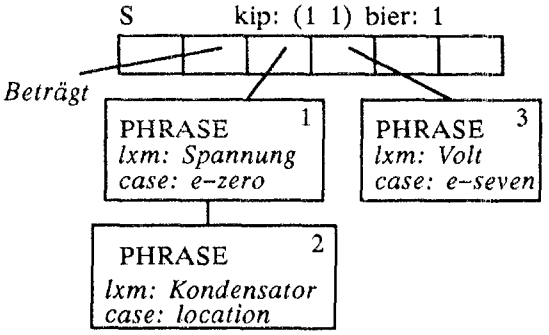
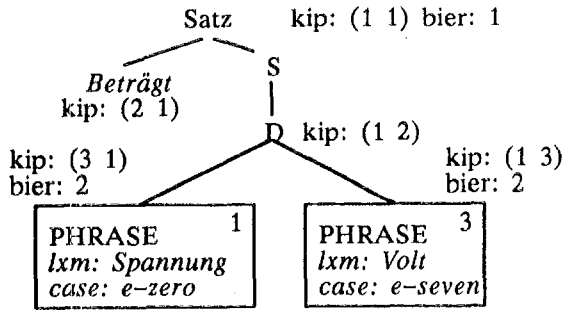


Fig. 5

Note that *beträgt* can already be synthesized, even though the rest of the syntactic structure has not unfolded yet. For algorithm (3), actually three nodes in Kiparski's notation are comprised in *S*: *Satz*, *S* and *D*. Therefore, for (3) the structure has to be viewed as if it looked like the one in fig. 6.

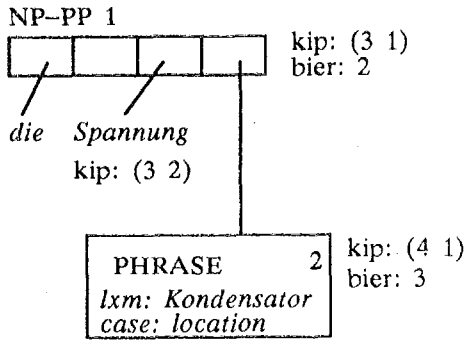
(3) applied to *Satz* yields the pair (1+1 1) for *beträgt* and (1 1+1) for *S* (tail stress). *S* has only one successor, therefore (3) does not apply. It does, however, apply to *D*, where the pairs (1+2 1) and (1 2+1) are computed for the two PHRASEs (tail stress). The Bierwisch index is simply incremented by 1 for both PHRASEs. Thus the string in the lower left of fig.6 can already be written (phonemes are given in an ASCII representation of IPA notation, stress markers are preceded by ", boundary indices by #).



#0 b$tr"2Egt #1

Fig. 6

The process now recursively continues by generating the left PHRASE (Kiparski pair (3 1), Bierwisch index 2). As above, a corresponding surface building block (NP–PP) is generated and filled with lexical items and the modifying PHRASE (*"am Kondensator"*). The structure so produced is shown in fig.7.



#0 b$tr"2Egt #1 dI #2 Sp"3an=N #2

Fig. 7

Algorithm (3) is applied once (tail stress) and yields a stress marker 3 for *Spannung*. The Bierwisch index is incremented once again for the nested PHRASE 2 (note that the Kiparski pair for that PHRASE is the same as for a loser although it is behind the 'tail'. Kiparski, in his original article, did not mention post–head modifiers). This PHRASE will subsequently be generated accordingly. The lower right of fig. 7 shows the result at this stage. The determiner *die* is not a stressable item and therefore does not receive a stress marker. The noun, on the other hand, is provided with the marker 3.

After the final lexical item of PHRASE 2, *Kondensator*, a boundary marker 3 will be written. Now the last part of (4) comes to bear. As it is the end of the phrase, it is overwritten by the marker of the dominating phrase (NP–PP 1), 2. It is also the end of NP–PP 1, so it is finally overwritten by the marker assigned to S, which is 1. The output at this stage is the following:

#0 b$tr"2Egt #1 dI #2 Sp"3an=N #2
Ham #3 k0nd$ns"4Ator #1

After that, PHRASE 3 – the next one attached to S – is generated, in an analogous fashion.

7. Discussion and Conclusion

The experiences with the described generator have shown that synthesis of German utterances in a concept–to–speech system is possible while both synthesizing intonation patterns using syntactic information and maintaining the efficient process structure of the generator designed for the specifics of the German language. The assumptions under which it was applied are a single–sentence system without contextual or pragmatic information. Problems rooted in the lack of such information have therefore not been solved. The speech produced this way shows considerable improvement over monotonous versions or versions which cannot make full use of syntactic information. Furthermore, the approach can easily be extended to include additional aspects of intonation such as emphasis of elements over others.

Despite the success of the system described in this paper, some limitations have been discovered. In the test domain long sentences with complex and multiply nested phrases were quite frequent. Some of them included post–head modifiers such as *"rechts unten"* (= "to the lower right"), in additon to other modifiers like several adjectives. The algorithm by Bierwisch produced boundary markers between the beginning and the end of *"rechts unten"* that were only slightly greater than the surrounding ones. Synthesis of the utterance, however, revealed that the modifier was spoken with an unnaturally high pitch and a pause that was too short. Manually altering the indices to lower values, which would mean that *"rechts unten"* is a constituent on sentence level rather than a noun modifier, lead to better results. Thus, the top–down scheme of the algorithm would have to be broken in this case.

Future work will be required to discover other limitations and to adapt the process to overcome them.

References

Bierwisch M.: Regeln für die Intonation deutscher Sätze. In: Studia Grammatica VII, Berlin, 3rd ed., 1973.

Chomsky N.: Reflections on language, MIT Press, Cambridge, MA, 1975.

Dorffner G., Kommenda M., Frenkenberger S.: Ein Oberflächengenerator zur Erzeugung geschriebener und gesprochener Sätze; Austrian Research Institute for Artificial Intelligence, Vienna, TR 88–10, 1988.

Dorffner G., Trost H., Buchberger E.: Generating spoken output for an expert system interface, ÖGAI–Journal 3–4, 36–41, 1988.

Engel U.: Syntax der deutschen Gegenwartssprache, 2nd ed., Erich Schmidt, Berlin, 1982.

Frenkenberger S., Kommenda M., Pounder A.: Automatische Wortklassifizierung und Prosodiebestimmung im Sprachausgabesystem GRAPHON; ITG–Fachbericht 105, Digitale Sprachverarbeitung, 1988.

Kiparsky P.: Über den deutschen Akzent. In: Studia Grammatica VII, Berlin, 3rd ed., 1973.

Kulas W., Rühl H.–W.: Satzzerlegung für ein Sprachausgabesystem mit unbegrenztem Wortschatz, Fortschritte der Akustik – FASE/DAGA'82, pp.1017–1019, 1982.

McDonald D.: Natural language generation as a computational problem; in: Brady & Berwick (eds.): Computational models of discourse, MIT Press, 1983.

Schnabel B.: Developpement d'un système de synthèse de l'Allemand a partir du texte, Thèse de doctorat, Université Stendhal, Grenoble, 1988.

Zinglé H.: Traitement de la prosodie en Allemand dans un système de synthèse de la parole, Thèse d'Etat, Université de Strasbourg II, 1982.