Dept. for Speech, Music and Hearing
Quarterly Progress and Status Report

A text-to-speech system based on a phonetically oriented programming language

Carlson, R. and Granström, B.

KTH Computer Science and Communication

journal: STL-QPSR
volume: 16
number: 1
year: 1975
pages: 017-026

http://www.speech.kth.se/qpsr

B. A TEXT-TO-SPEECH SYSTEM BASED ON A PHONETICALLY ORIENTED PROGRAMMING LANGUAGE *

R. Carlson and B. Granström

Abstract

The general philosophy of a synthesis system can be described in a straight-forward fashion, but when more detailed information is needed it is harder to obtain, since it is usually hidden in the form of a programming language far away from linguistic and phonetic rule writing conventions. A new type of programming language should solve some of these problems by making it conform as closely as possible to phonetic and linguistic terminology. Such a language is presented in the first part of our paper. In the second part of the paper we give an example of how this tool is used in a rule synthesis system for Swedish. This system is made to use orthographic input and does accordingly consist of rules describing graph to phoneme and phoneme to phone conversions as well as prosodic realizations. Punched paper tapes used by printers for type setting have been used as input to the system with promising result. However, a number of corrections and expansions are yet to be made in order to improve the speech quality and the graph to phoneme transformation process.

---

* This is an expanded and revised version of a paper presented at the Speech Communication Seminar, Stockholm 1974: "A phonetically oriented programming language for rule description of speech"

Introduction

Several synthesis systems have emerged during the past years and are still being developed. A synthesis system can generally be split into three parts: a) a library store for segmental units and their associated variables and features; b) some method for changing these units into positionally dependent segments; c) an output routine in which the finally selected variable values control either an articulatory model or a terminal analog. The output from the articulatory model could either be acoustical or formant-coded.

The general philosophy of such a system can be described in a straight-forward fashion but when more detailed information is needed it is harder to obtain since it is usually hidden in the form of a programming language far away from linguistic and phonetic rule-writing conventions. This means that a change of the system on account of linguistic and phonetic facts presupposes a programming effort, the amount of which depends on the programming language and the complexity of the synthesis program actually used. Thus seeking a balance between the uncovering of natural language structure and developing of synthesis strategies might be extremely complicated or even impossible due to limitations in the system used.

A new type of programming language should solve some of these problems. We have tried to define such a language and make it conform as closely as possible to phonetic and linguistic terminology. The function of the synthesis system should accordingly be explained in an explicit fashion in the sequence of rules written in this language.

In the second part of the paper a preliminary text-to-speech system is presented as an example of how this tool is used.

Basic rule structure

In this section we shall describe the kinds of rules that define any specific synthesis program. As will be seen, these rules could be regarded as variations of one single rule structure close in form to the ones used in generative phonology but with some important generalizations.

The rules work on string elements and these elements must be defined by the user. The definition can include a specification of distinctive features and variables associated with a string character. We have provisions for up to 32 basic features that could be labeled according to the user's choice and whose values can be specified as plus or minus or zero giving, in fact, a ternary opposition. The basic features can also be grouped into any number of secondary features giving a handy description of natural classes. The variables used must also be labeled and they can be either one- or two-dimensional. The two-dimensional ones can have more than one value associated with an element. The latter kind of variables are typically intended for time/value or space/value description. There is no inherent restriction on the number of possible variables.

The basic structure of our rules is:

(1) CRULE X $\longrightarrow$ Y/A & B

where & marks the place where the structural description X occurs in the context description A B. Y denotes the structural change and $\longrightarrow$, and / are mere delimiters. The rule name CRULE means that this rule applies only once in a certain position of the string. If we want recursive application of the rule we write

(2) LCRULE X$\longrightarrow$ Y/A & B

and the rule will be looped until no further application is possible.

In generative phonology there is a wide usage of the braces convention that is a means of collapsing of rules with disjunctive ordering, i.e. if one rule in the group is applied, the following rules are disregarded. Since we want to keep to "on-the-line" expressions we write the rule separately but end with @ as seen in (3).

(3) CRULE X $\longrightarrow$ Y/A & B @

meaning that this one and the next rule is part of a disjunctively ordered set of rules.

In (1) - (3) A, B, X, Y stand for strings that could all be empty. When A and B are empty we simply write

(4) CRULE X $\longrightarrow$ Y

i.e. a context-free rule,

X being empty implies an insertion and when Y is empty a deletion takes place.

The strings A, B, X are strings of elements and could contain string characters but also more weakly or more strictly specified elements within < > parentheses, e.g.

(5) A = a <+KONS, -VOICE> <i, STRESS > 3, x:=F1>

The < > parentheses could thus contain variable and feature conditions on the context and also one string character. Here x:=F1 means that the external variable x is given the value of the parameter F1 in this element and could be used in the structural change. The inclusion of optional elements in the string makes compact rule descriptions possible. In this formalism such an operation is designated by a (,) notation, e.g.

(6) < +KONS > (1, 3)

specifying the length of a consonant cluster of minimally one and maximally three segments. A blank or zero in the first position means none and in the second position a blank means an infinite number.

The structural change Y in (1) - (4) is a string of elements similar to the strings X, A, B but here the < > parentheses express the result of a change like

(7) Y = <a, +ACCENT, F1=N >

An element will thus be changed to one corresponding to the character "a"; the feature +ACCENT will be added to its feature specifications and the value of the variable F1 will be altered to N.

If a value is to be inserted, a ":=" is used instead of a "=" and if a value is to be deleted a "=#" is used. N is a numerical expression of arbitrary complexity where internal variables from the element itself and external variables as x in (5) could be used with standard numerical functions.

The system in operation

This section contains a brief description of how to use the system, while the following section will give a specific example of rule synthesis. Fig. I-B-1 shows a block diagram of the system.

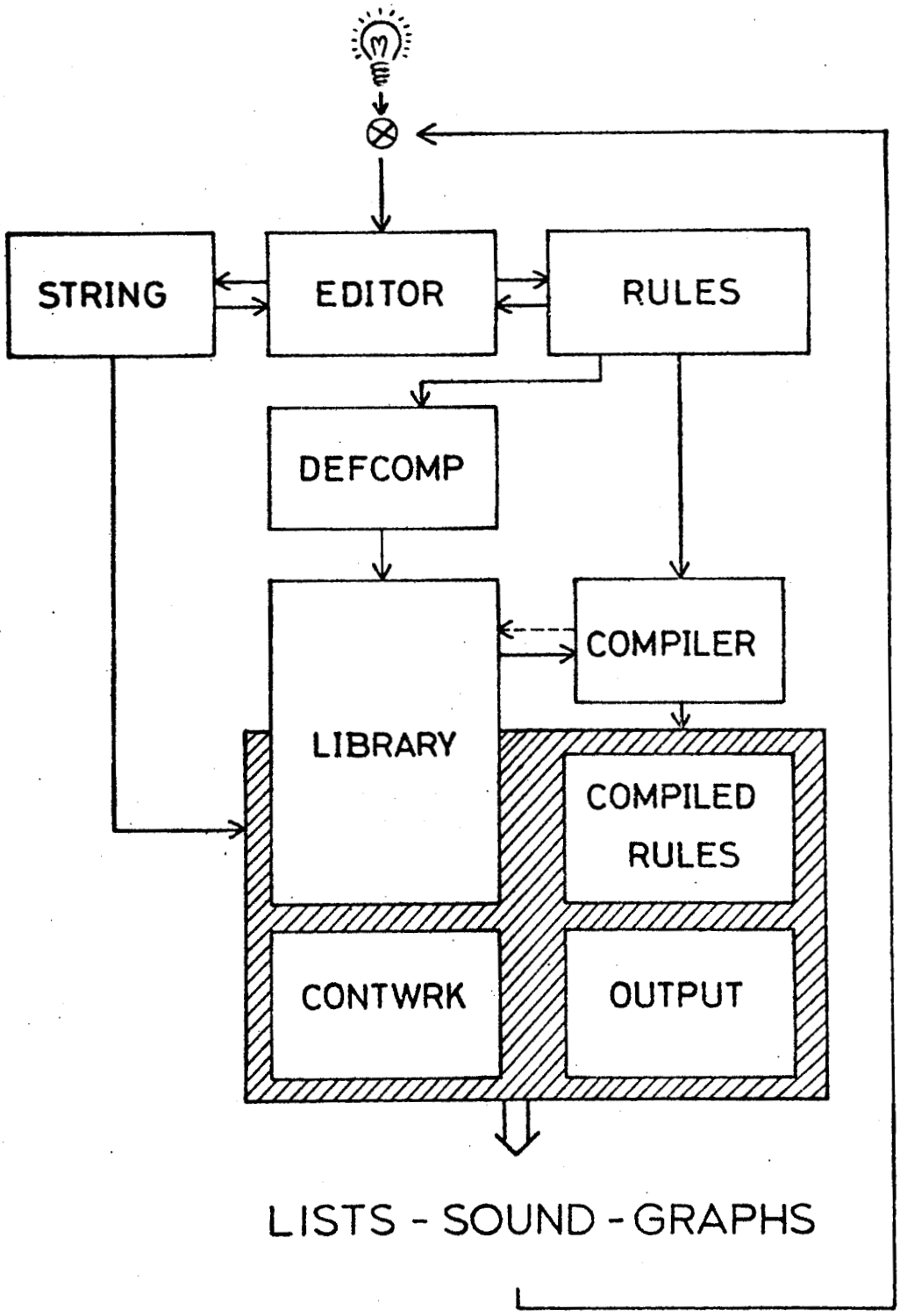

Fig. I-B-1. Block diagram of the system program.

To start from the beginning the user gets an idea of how the synthesis is to be made. This idea is formulated into rules according to the description in the preceding section. These rules can be edited in the program editor and stored. The user must also define the string elements he is going to use as well as features and variables associated with these. This definition rule system is run through the definition compiler (DEFCOMP), which will build up the required library. No more feature or variable names are now allowed to be used in the rule system except the ones which occur in the library. The library could easily be changed at any time by running a new definition rule sequence through DEFCOMP. The ordered synthesis rules should now be compiled by the compiler and will then be stored as instructions in a machine language.

The system is now ready to work on a specific input text, the string. This is done by the block CONTWRK in accordance with the library and the compiled rules. The result is stored on a disc-store as an output source to be used by the OUTPUT-ROUTINE. OUTPUT presents the result either as lists and graphs or as analog and digital voltages. Some type of hardware or software synthesizer is usually connected to some of these outputs. The OUTPUT- routine could make use of a variable in the output source in several different ways. The variable could be smoothed by a smoothing subroutine before presentation and the smoothing could be changed by a separate variable, i.e. as an output manifestation a variable could change the output routine of another variable. Two variables could be mixed together by some mathematical formula and used as a single output. All these kinds of manipulations are set by the operator and could easily be expanded for special purposes.

Special system rules could be added to the rule system giving the operator the possibility to examine the result of rule applications at different stages in the processing of the string. At any point in the rule sequence comments (C....) and break points (BREAK) could be inserted. When a BREAK occurs the operator can examine the status of the currently modified string as well as the status of the program, rules, and library. The OUTPUT routine could also be used at this point both for listening tests and for plotting of graphs.

A special type of rule gives the possibility for optimizing the system and for production experiments. By means of a joy stick external variables could be introduced by the operator at this point. The joy stick coordinates are displayed on a screen for visual feedback. These variables could be used in the following rules resulting in any kind of change. The program could be looped at this point until the operator is satisfied. This method has been used in several experiments, one of them presented in another contribution to this issue of STL-QPSR/3/.

Each time a rule is applied a counter in the rule is incremented by one. By this method the productivity of each rule can be statistically evaluated.

A preliminary text-to-speech system

The general approach

As an example of how the program language is used in practice we will give a brief description of a text-to-speech system for Swedish. Most existing orthography-to-speech systems (e.g. systems developed in USA at Haskins Laboratories, Bell Laboratories, and Massachusetts Institute of Technology) utilize an extensive word or morph lexicon that necessitates from ten thousands to a couple of hundred thousands of items stored. Our approach aims at a system where instead we try to minimize the lexicon and make the inventory of rules more exhaustive. Most of these rules are necessary anyhow to cover items that are not in the lexicon. The number of errors will perhaps be higher in such a system but still every word will be pronounced according to the rules of the language in a manner less disturbing than the spelling out solution (Haskins). The performance of the complete systems will be the ultimate test of the acceptability of the different approaches.

The design of the system

Since the rules apply in order the system can roughly be illustrated as a hierarchical build-up, see Fig. I-B-2. The labels on the boxes must be interpreted with some caution as will be seen later. The first box is a lexicon but has the dual function of taking care of exceptions and marking common function words.

The classification of a word as a function word is the primary argument for storing the word in the lexicon. And so far only about 100 words

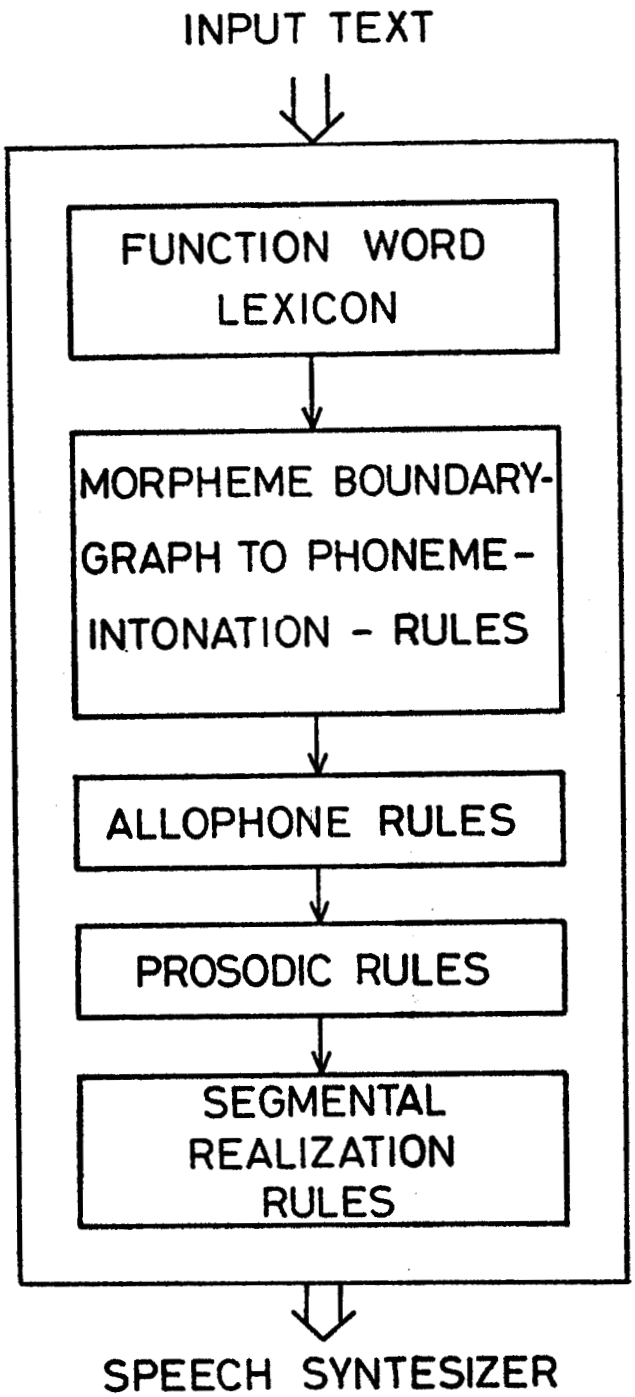

Fig. I-B-2. The hiearchical build up of the text to the speech system.

have been incorporated. The lexicon is simulated by a sequence of rules but will in the future be taken care of in a more rapid and straight-forward fashion. An example is given below.

(8) CRULE VAD ⟶ VA: + / < -SEG > & < -SEG >
(VAD = Eng. What)

The D is deleted and the tense vowel quality is chosen inspite of lack of stress. The final + marks a tendency for VAD to group with the following words. This information will be utilized in the prosodic rules.

The next stage is the graph to phoneme conversion. Here the ordering of rules is essential as can be seen in a simplified rule sequence describing part of the generation of the Swedish [ç] and [ɧ].

(9) CRULE K ⟶ TJ / < -SEG, WB > & < +VOK, +MJUK >

(10) CRULE KJ ⟶ TJ

(11) CRULE STJ ⟶ SJ

	(9)	(10)	(11)	
KELA	TJELA			[ç]
KOLA				[k]
KJOL		TJOL		[ç]
SJÄLV				[ɧ]
STJÄLK			SJÄLK	[ɧ]
SKELA	STJELA		SJELA	[ɧ]
SKOLA				[sk]
SKJUTS		STJUTS	SJUTS	[ɧ]

The feature < +MJUK > ≈ soft is part of the feature specification of vowels causing [g] / [j] and [k] / [ç] alternations but the feature also governs the Umlaut. WB stands for word boundary.

If, for example, rule (11) came before (9) false forms like [sç ets] and [sç ela] instead of [ɧets], [ɧela] should be produced.

The rule system above works on the normal graphic representation. KJ can only occur in morpheme initial stressed position, i. e. the KJ can be used for morpheme boundary setting and stress marking. Derivation of word tone and vowel quantity constitutes a special problem in Swedish. These problems are best dealt with in conjunctions with the graph to phoneme conversion.

For example, compounding can be detected both by unusual consonant sequences and by vowel quality. The Swedish Å, Ä, and Ö have to be stressed in most positions. Other stressed vowels are easily found in affixes or before double-spelled consonants. By different stress-assignment rules several vowels could have primary stress in a word and this is of course not possible. In that case we have a probable compound where the first stressed vowel retains its primary stress and the last is given secondary stress. Tense vowels with reduced stress because of compounding keep their tenseness which should be kept in mind when the tense marking rules occur.

To summarize, the graph to phonetic transformation is made by an ordered mixture of different kinds of rules, such as graph to phoneme, stress, accent, and compounding rules.

The graphic to phonetic transformation by rules will always make errors because of irregular words, mostly foreign words, or an unsufficient rule system. The last could be corrected with help of a deeper understanding of the problem, the first could not.

The following group is the allophone rules. The distinction between extrinsic and intrinsic allophones is a matter of discussion but in this group we include the rules that could apply before the prosodic realization rules. One example is a rule causing deaspiration of voiceless plosives.

(12) CRULE < -CONT, -VOICE, +KONS > → < +VOICE> /
< +OBST, -VOICE, +KONS > &

+VOICE might seem a peculiar structural change but it will be justified in rule (14). The prosodic rules in the next stage incorporate the fundamentals of the prosodic model of Swedish described elsewhere /2, 4, 5, 6/. One example is a modified segment duration rule

(13) CRULE < +SEG > → < DR:=T * (A+B+6) / 10 * EXP(-LOG(B)
*0.12-LOG(A) * 0.35)>

where T is a nominal length and A, B, are variables depending on the position and "word" length. In this group also segmental junctural marking like glottalization takes place.

The box marked segmental realization rules takes account of allophonic variations often depending on the output of the prosodic rules like:

(14) CRULE < +SEG > ⟶ < TTENSION=TTENSION+X/2 > /< -CONT, -VOICE, +KONS, X:=DR > &

TTENSION means the time for the change of the tension parameter. This parameter corresponds roughly to glottal closure[7.8]. The glottal closure, is delayed after a -VOICE stop, causing aspiration (cf. rule (12)). Another rule is the diphthongization of some high vowels, for instance:

(15) CRULE < +VOK, +HIGH, +BACK, +TENSE, +STRESS > ⟶ < LF1:=270, TLF1=DR/2, LF2:=F2 * 0.85, TLF2 = =TLF1, LF3:=2050, TLF3=TLF1 >

Here we insert new points for the formant F1-F3 causing a final labialization gesture in long stressed back vowels. (TLF1 means Time for Last F1.)

The derived parameters are then ready to control, in this case, a terminal analog synthesizer via the OUTPUT module.

Work in progress

Work is now under way to optimize the text-to-speech system at different levels. This is done by corrections, insertions, and reordering of rules. Test sequences of different kinds like VCV lists and short stories are used in listening tests. For the graph to phoneme transformation we use the most common 10.000 words in Swedish[1]. When the words have been processed the transcriptions are automatically compared to stored "correct" transcriptions with help of a computer program. The work on the system has been simplified to a high degree by this automatic procedure. It should be noted that even if an "uncorrect" transcription is given, many words are accepted by listeners, especially in running synthesized speech.

Final comments

The programming language described has been used in a number of applications including the outlined preliminary version of an orthography to speech system. As yet we have noted no severe built-in limitations. Some extensions are of course possible and might add to the usefulness

of the system. Some of the extensions are typical of the program in the present environment like output routines, some might speed up the execution like special lexicon look-up rules but some are of a more fundamental nature like one to many generations of strings. The latter feature should be useful in automatic speech recognition (ASR) when the problem is either to make well formed strings out of an imperfect input string according to the rules of language or to predict possible pronunciations of a hypothesized utterance.

One of the main advantages of the programming language appears to be that the "knowledge" in the system is explicit in the print-out of rules that are legible to most people working with assocated problem areas in language and speech research.

The text-to-speech system is meant to function as a reading machine for the blind as well as an audioresponse equipment using an unrestricted vocabulary. Informal tests using punched paper tapes as input have already given promising results. We feel that such a machine could be based on rules at all levels with a small dictionary containing mostly function words.

References:

(1) Allén, S.: Nusvensk frekvensordbok, 1 (Frequency Dictionary of Present-Day Swedish), Almqvist & Wiksell, Stockholm 1970.

(2) Carlson, R. and Granström, B.: "Word accent, emphatic stress, and syntax in a synthesis by rule scheme for Swedish", STL-QPSR 2-3/1973, pp. 31-36.

(3) Carlson, R. and Granström, B.: "Perception of segmental duration", this issue of STL-QPSR, pp. 1-16.

(4) Carlson, R., Erikson, Y., Granström, B., Lindblom, B., and Rapp, K.: "Studies of the rhythm and intonation of Swedish", to be publ. in Speech Communication, Vol. 2; Almqvist & Wiksell, Stockholm 1975.

(5) Carlson, R., Granström, B., Lindblom, B., and Rapp, K.: "Some timing and fundamental frequency characteristics of Swedish sentences: data, rules, and a perceptual evaluation", STL-QPSR 4/1972, pp. 11-19.

(6) Lindblom, B., and Rapp, K.: "Some temporal regularities of spoken Swedish", PILUS 21, Stockholm University (1973).

(7) Rothenberg, M.: "Glottal noise during speech", STL-QPSR 2-3/1974, pp. 1-10.

(8) Rothenberg, M., Carlson, R., Granström, B., and Lindqvist-Gauffin, J.: "A three-parameter voice source for speech synthesis", to be publ. in Speech Communication, Vol. 2; Almqvist & Wiksell, Stockholm 1975.