COMPUTATIONAL PHONOLOGY: MERGED, NOT MIXED

Egon Berendsen, Department of Phonetics, University of Utrecht, The Netherlands
Simone Langeweg, Phonetics Laboratory, University of Leyden, The Netherlands
Hugo van Leeuwen, Institute of Perception Research, Eindhoven, The Netherlands

0. Introduction

Research into text-to-speech systems has become a rather important topic in the areas of linguistics and phonetics. Particularly for English, several text-to-speech systems have been established (cf. for example Hertz (1982), Klatt (1976)). For Dutch, text-to-speech systems are being developed at the University of Nijmegen (cf. Wester (1984)) and at the Universities of Utrecht and Leyden and the Institute of Perception Research (IPO) Eindhoven as well. In this paper we will be concerned with the grapheme-to-phoneme conversion component as part of the Dutch text-to-speech system which is being developed in Utrecht, Leyden and Eindhoven.

One of our primary interests is that the grapheme-to-phoneme system not only has to generate the input for speech synthesis, either in allophone or diphone form, but that it had to be used for other purposes as well. Thus, the system has to satisfy the following demands:

- its output must form a proper and flexible input for diphone as well as allophone synthesis;
- it must be possible to easily generate phonematized lists on the basis of orthographic input;
- it must be possible to automatically obtain information regarding the relation between graphemes and phonemes in texts;
- the system has to be user-friendly, so that it can be addressed by linguists without computer training (for example to test their phonological rules).

In our view, there are two aspects to a grapheme-to-phoneme conversion system: a linguistic and a computational one. The linguist, in fact, provides the grammar necessary for the conversion and the engineer implements this grammar into a computer system. Thus, knowledge about spelling and linguistics are **separated** from the technical implementation: the linguist provides the rules and the system executes them. The two components will also constitute the main sections of this paper.

1. Linguistics

For grapheme-to-phoneme conversion it is expedient to assume several modules. In the first module, 'difficult' elements like numbers, acronyms and abbreviations, have to be changed into their corresponding full graphemic notation. Next, one has to recover units from the spelling which influence grapheme-to-phoneme conversion: for example, words forming compounds are written as one uninterrupted string in Dutch, but have to be recovered because they influence graphemic conversion and stress assignment. The third and most important module concerns the rules which assign phonemes to graphemes. We then have phoneme information, and can establish further relevant units on the basis of this information. Finally, phonological processes have to be accounted for in modules for stress assignment and segmental phenomena.

1.1. Phoneme-to-grapheme assignment

Our starting point for the development of the grapheme-to-phoneme system is that graphemes and phonemes are different entities, which should be represented at separate levels. As graphemes form the input for the conversion, the grapheme level is filled from the start. The derivation of phonemes is performed in the following way: to each grapheme or group of graphemes a corresponding phoneme is assigned at the phoneme level. This is represented in (1), where lower case letters indicate graphemes and capitals indicate phonemes.

(1) grapheme level: a b c d e f g
phoneme level: A B K

Notice that the assumption of two levels, one for graphemes and one for phonemes, makes it possible to obtain information about the relation between graphemes and phonemes. As we will see below this assumption has some other attractive consequences.

Notice furthermore, that to each grapheme or sequence of graphemes a phoneme has to be assigned, except in cases where there is no corresponding phoneme. For the non-linguist, it may come as a surprise that a great number of regularties and subregularities in correspondance between graphemes and phonemes can be found. The assignment of phonemes to graphemes is therefore done by rule. Of course, there are always words that cannot be captured by the rules: these will have to be enumerated. The order of application will then be enumeration, sub-rules and rules. The string of graphemes is scanned sequentially: first a phoneme is assigned to the initial grapheme, then this is repeated for the second grapheme and so on. This procedure works very quickly since, if the grapheme under consideration is an **a**, only the rules assigning phonemes to **a** will have to be considered. The rule format used here, is very similar to the well-known format of Chomsky and Halle (1968) (=SPE). Some further mechanisms are added to their rule format:

- it is possible to negate elements or groups of elements in the environment of the rule;
- one can use so-called global rules, referring back to information available earlier in the derivation because we use a two-level approach;
- it is possible to use definitions instead of repeatedly used sequences, in the rule environment, such as sequences indicating syllable boundaries.

Below, we will demonstrate how our linguistic knowledge can be expressed in rules and sub-rules. These rules will not cover all cases, but they will serve as an illustration. A rule completely written in the standard SPE format may be as follows.

(2) c,h -> SJ / _ i,n,{<-cons>}
{c }

This rule assigns the phoneme **SJ**/[š] to the letters **ch**, if the latter is followed by the letters **in** followed by either a vowel or **c**, thus accounting for **chinchilla** (chinchilla) and **China** (China). As one can see from (2) no "long" braces are used to indicate several alternatives, but each alternative is surrounded by "short" braces, since it is impossible to use "long" braces in the computer. Furthermore, phonological features are surrounded by angled brackets.

The possibility of negating an element is illustrated in (3).

(3) c,h -> SJ / i _ e,'1

The grapheme preceded by a quote in (3) is negated. This means that every sequence consisting of **ch** preceded by **i** and followed by **e** which in turn may not be followed by **1**, is assigned the phoneme **SJ**. Thus we account for the alternation in **ch**-pronunciation in **fiche** (chip) where **ch** is **SJ** and **richel** (sill) where **ch** is **X**/[x].

As a last example, a rule is given in which only phoneme information has a triggering effect.

(4) au -> OO / SJ _

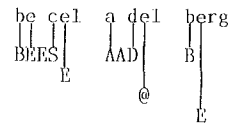
To the graphemic sequence **au** the phoneme **OO**/[o:] is assigned if it is preceded by the phoneme **SJ** as is the case in **chauvinisme** (chauvinism) where **ch** is **SJ**. Notice furthermore, that one can only use phoneme in-

formation in the lefthand environment of rules assigning phonemes to graphemes, since the string is scanned from left to right and grapheme by grapheme.

A further addition to the SPE format which has been developed is the possibility to require two things at the same time in the environment of rules: so-called coordinations. The need for such an option is illustrated by the following example. Using standard SPE rules, we have to postulate the two disjunctively ordered rules in (5) plus the rule in (6) to account for the variation in pronunciation of final **e** in **Becel** (e=E/[ɛ]) (trade mark) and **adel** (e=@/[ə]) (nobility). Rule (6) is independently motivated by cases such as **berg** (mountain). Rule (5a) takes precedence over rule (5b) and (5b) over (6). This means that if (5a) has been applied both (5b) and (6) will not be applied, although their requirements are met. Notice that in (6) the possibility to use definitions is illustrated.

(5)a e -> E / c _ 1,<-segm>
b e -> @ / VOC,CONS _ 1,<-segm>
(6) e -> E / _ SG
where SG constitutes the definition for a sequence of consonants which follow a vowel in a closed syllable

The derivation of the examples mentioned above is as follows.

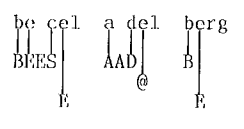
(7)


(5a)
(5b)
(6)

However, if we have the possibility to use coordinations, we only have to state rule (8) which takes precedence over rule (6). The + symbols placed beneath each other indicate coordination.

(8) e -> @ / VOC,+CONS _ 1, <-segm>
+'c

The derivation (7) now turns into (9).

(9)


(8)
(6)

1.2. Other modules

Until this point in the paper, we have concentrated solely on grapheme-to-phoneme conversion proper, but as we already stated above, the block of rules assigning phonemes to graphemes is surrounded by other rule modules with different functions. We will deal with some of these modules here, disregarding abbreviations, acronyms, numbers, and phonological processes above the word level.

Since Dutch is a stress language, one aspect of the grapheme-to-phoneme conversion must be the assignment of word stress. The question is then how word stress must be assigned. Dutch word stress is not fixed, i.e. always assigned to the same syllable position, but lexical, i.e. the position may vacillate (cf. **kálium** (potassium), **kaboúter** (imp), **kapitein** (captain)). The rules then must refer to morphological and/or syllable structure.

Syllable weight is decisive in stress assignment in monomorphemic words. For compounds, however, morphological structure has to be recognized. By making reference to sequences of vowels and consonants, the syllable weight can be defined. As is well-known from SPE, stress rules in monomorphemic words are disjunctively ordered (cf. (10)). For our system, the implication is that the whole input string (for these stress rules) has to be scanned for each sub-stress rule. Furthermore, <-st(ress)> has to be present as a feature of the vowels in the righthand environment of the rule. In most cases, compound stress is assigned to the first word of the compound. Within the SPE-format, compound stress assignment leads to stress lowering of the stresses that have already been assigned. Our rule system, however, assigns secondary stress in monomorphemic words which is then raised to primary stress by an additional rule in both monomorphemic words and the first part of a compound (cf. (11)). The number following a feature indicates that this item must be present at least that number of times.

(10) VC = +<+voc> ! So @ cannot be stressed
+'@
a) <+voc,+long> -> <2st> / _ CONS,#
VC -> <2st> / _ CONS2,#
VC -> <2st> / #,CONSO _ CONSO,#
b) VC -> <2st> / ...
... _ CONSO,+II ,CONSO,+<+voc,-st>,CONSO,#
+<-st> +'II
+'@
c) VC -> <2st> / ...
... {<-st>},CONSO _ CONSO,<+voc,-st>,CONSO,#
{# }
(11) <2st> -> <1st> / #,#,<+segm>0 _

Therefore, before assigning phonemes to graphemes, the grapheme string has to be changed in such a way as to mark those affixations and compoundings that influence this assignment and the subsequent phonological operations. The operations in this module are also applied by rule, in the format already dealt with. This approach is not obvious from the outset. However, the affixes influencing the conversion and the phonology are limited and as such, they can be recognized by rule. Furthermore, many compounds have internal grapheme sequences which do not occur in monomorphemic words. On the basis of these sequences, boundaries can be inserted. Of course, this leaves us with compounds which do not have clusters which are impermissible in monomorphemes. An exception lexicon will then be necessary.

Consider the following example. In Dutch, a sequence of an obstruent followed by a voiced obstruent hardly ever occurs in monomorphemic words (cf. Zonneveld (1983)). This characteristic is also represented in the spelling. Thus, a rule of the form in (12) separates parts of compounds.

(12) Ø -> # / <-son>i _ <-son,+voice>j,CONSO,VOC
where **i** and **j** indicate different segments

After application of rule (12) **huisdeur** (front door) is represented as **huis#deur**. In fact, this rule is somewhat more complicated by using coordinations, since we also have to exclude overgeneralizations such as **huis#den** (lived). The graphemes and boundary in **huis#deur** are assigned phonemes by conversion rules in the next module, furthermore the stress rules dealt with above, assign stress and finally the phonological rule of regressive voicing assimilation applies, converting **S** into **Z**. The separate stages of the derivation of **huisdeur** are shown in (13).

(13) input: h u i s d e u r
boundaries: # # h u i s # d e u r # #
conversion: # # H UI S # D EU R # #
sec. stress : # # H 'UI S # D 'EU R # #
prim. stress : # # H 'UI S # D 'EU R # #
phonology: # # H 'UI Z # D 'EU R # #

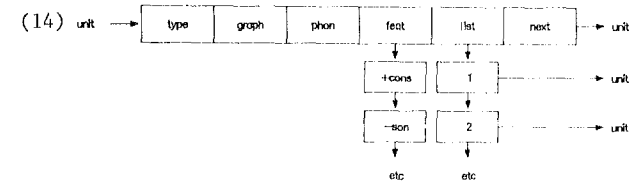
2. Implementation

Now that the linguist has provided the rules it is necessary to consider these from a technical point of view. In these linguistic rules one is able to describe the phoneme-to-grapheme assignment in terms of graphemes, phonemes and their context. In principle, there are two basic possibilities to do so. Either one refers directly to a basic entity (e.g. grapheme or phoneme, in which case the structure has a fixed length) or one uses a larger structure that describes the context in a more complex manner. A basic entity can be referred to explicitly, by stating the grapheme or phoneme involved, but also implicitly, by specifying some features to define a set of phonemes or graphemes for which the context is valid. As to the structures, several are available. The first one

is alternative validity: one of the specified structures must be valid to produce a match (cf. rule (2)). The opposite structure is simultaneous validity: all of the specified structures must be valid to produce a match (cf. (5)). A third possibility is negated validity: when the structure is valid, no match will result and vice versa (cf. (3)). A fourth (commonly used) structure is optional validity: the specified structure may or may not be present (cf. (12)).

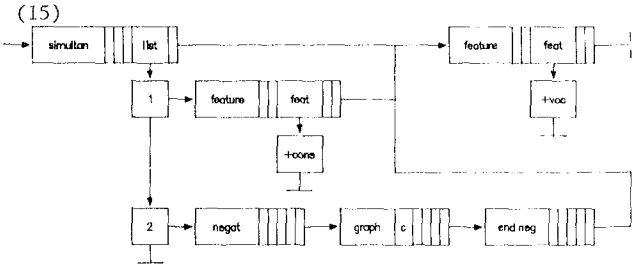
It will be clear that the system gains considerably in power by allowing combinations of these structures. For the implementation this has some non trivial consequences. First of all, a suitable data representation must be found to store the linguistic knowledge. It is desirable to have this representation in a compact and efficient form, as it will be consulted 'on-line' during the transcription process. Because there are no restrictions on the use of these structures, the use of a dynamic data structure to represent the knowledge seems appropriate in order to prevent too much a waste of memory seems obvious. This dynamic data structure consists of a variable number of linked units. Each unit consists of a number of 'fields' to represent the different types of information one needs. This is illustrated below.

(14)



In the field **type** an indication is given of the type of information and its location. Next, there are four fields, one of which contains the linguistic information formulated in the rules. If the unit is a grapheme, the grapheme concerned is stored in the field **graph.** As a phoneme is seen as a different entity, a phoneme will be represented in the next field, denoted by **phon.** Then it is possible to use features to indicate a set of phonemes or graphemes. These are of two different types but are stored in the same field **feat.** This actually is a pointer to a list of features, containing the value (+ or -) and the feature concerned. The final information field contains a list of other units. This is needed to represent the alternative or simultaneous validity. Both of these structures are stored in this list, and in combination with the first field containing the type, the system knows how to interpret this list. The optional validity can be seen as an alternative validity: either the structure is present or not, and this is therefore represented as an alternative validity. Finally certain types do not contain any information, but are used as markers, i.e. **negation** and **end negation** which denote the beginning and end of a negated structure. Following these information fields, a last field **next** will refer to a following unit, which describes the next part in the linguistic rule. As an example of how this data structure represents linguistic rules, the lefthand context of rule (8) is shown in (15). Since the focus is the starting point, the data structure is constructed from right to left.

(15)



An input string will now be transcribed by comparing it with the appropriate units. While consulting this data structure an unexpected problem arises. Because of the freedom the user has to combine different structures, it is possible to build a structure which has different lengths (number of units) for different paths. The unit or structure following this variable length structure no longer has a fixed position with regard to the starting point. This may especially create problems when this first structure is negated as well. This can be explained best with an example which is hypothetical. The structure, however, could easily be used by a linguist and thus the system should be able to handle it correctly. Suppose the right context of a linguistic rule looks as follows.

(16) _ '{a },t
{o,u}

The rule states that it is not permitted that either an **a,** or an **o** followed by an **u,** is present before a **t.** The question is what exactly is meant with this negation. A negation is only meaningful within a closed set, and therefore the set is defined implicitly by the unit or structure being negated. **'a** (not **a**) means: all graphemes except **a.** **'[o,u]** means: all sequences of two graphemes except the sequence **ou.** The sequence **at** will therefore belong to this second set. If the input string now consists of **att,** the first path will reject the string, but the second path will approve of it. As both paths must approve of the string to produce a match, this string will be rejected. However, it is insufficient only to look at the negated part (and then when no match is detected, consult the positive part). An input string **art** would then be rejected on account of the leading **a,** which would be incorrect. As there is not **t** directly following the **a,** the first path can give no verdict on the string and should pass it to the second path which would approve it. It is therefore necessary to consider all paths in combination with all following units. For further discussion of this particular problem, we refer to Van Leeuwen et al. (1986).

3. Concluding Remarks

In this paper we have dealt with the system for grapheme-to-phoneme conversion in Dutch as it is being developed at the Universities of Utrecht and Leyden and IPO, Eindhoven. We have shown that knowledge about spelling and phonology provides a proper grammar for automatic phoneme-to-grapheme assignment and that linguistic rules can be implemented without ad hoc mechanisms. Speed was considered an important performance feature in constructing the database as well as in consulting it. Typical values are: 20 seconds to (re)construct a new database (for instance for testing new rules or new versions of rules), and some 25 grapheme-to phoneme conversions per second. This phoneme-to-grapheme assignment system has been linked to the diphone speech synthesis system that has been developed at IPO. At the moment, the system is being tested on a lexicon of about 4000 monomorphemic words.

References

CHOMSKY, N. AND M. HALLE (1968), **The Sound Pattern of English.** Harper and Row, New York.

HERTZ, S.R. (1982), From text to speech with SRS. In: **JASA** 74, pp. 1155-1170.

KLATT, D.H. (1976), Structure of a phonological rule component for a synthesis-by-rule program. In: **IEEE Transactions on Acoustics, Speech, Signal Processing** ASSP-24, pp. 291-298.

LEEUWEN H. VAN, S. LANGEWEG AND E. BERENDSEN (1986), 'Linguistics as an Input for a Flexible Grapheme-to-Phoneme Conversion System in Dutch'. In: **Proceedings of the IEE Conference on Speech Input/Output; techniques and applications,** pp. 200-206.

WESTER, J. (1984), SF: Contouren van een toegepaste fonologie. In: **De Nieuwe Taalgids** 77, pp. 30-43.

ZONNEVELD, W. (1983), Lexical and Phonological Properties of Dutch Voicing Assimilation. In: M.v.d.Broecke, V.v.Heuven, W.Zonneveld (eds.), **Sound Structures; studies for Antonie Cohen.** Foris, Dordrecht.