PRELIMINARY WORK ON COMPUTER TESTING OF A GENERATIVE PHONOLOGY OF DANISH

Hans Basbøll and Kjeld Kristensen[1]

1. Introduction

The purpose of the project to be reported here is to test a part of a generative phonology of Danish (which had been worked out by one of the authors (HB) before this project started). An abstract phonological representation (i.e. a systematic phonemic representation) of a given word is used as input, and the program together with different sets of "background data" (see below) changes this input form to one or several phonetic representations. These phonetic representations can then be compared to actual pronunciations of the input word. If there is no agreement between the computed phonetic representation and the corresponding actual pronunciation, then there must be some error, either in the input form or in the background data (or in the program, of course). This error should then be found and corrected. Thus the ultimate purpose of this project is to improve the part of a Danish phonology being tested. It is evident that this method only allows finding lack of observational adequacy, i.e., it can be used neither as a proof of observational adequacy, nor for finding lack of descriptive adequacy (in the Chomskyan sense).

The project was initiated at the third Scandinavian summer school for research workers in the field of computional linguistics, held in Copenhagen from July 29th to August 10th,

---

1) Kjeld Kristensen is an engineer and cand.phil. (in Danish). He is a teaching assistant at the university of Copenhagen.

1974.[1] The program has only been tested with a very small sample of (quite arbitrary) data, but its general structure may nevertheless be briefly reported here. A more detailed account is planned for the forthcoming volume of ARIPUC.

2. Strategy

The program together with the three sets of background data (see section 2.1 below) is called the "grammar". The grammar may be considered to consist of a linearly ordered set of phonological rules together with the principles governing their application (see section 2.3 below). The input form of a given word is first submitted to rule no. 1 and is possibly changed by this rule. The output form from rule no. 1 is the input form to rule no. 2, etc. The output form from the last rule is the output of the grammar (see section 2.5).

2.1 General organization of program and data

As shown in fig. 1, the grammar consists of the MAIN PROGRAM together with three sets of "background data", viz. UNITMATRIX, RULEMATRIX and RULEINDEX. The main program operates with integers corresponding to the different vowels, consonants and boundaries. Thus we need a subroutine which translates a string of IPA-symbols (the input to the grammar) into the corresponding string of integers, and another subroutine which translates a string of integers into the corresponding string of IPA-symbols (the output from the grammar), see section 2.5 below.

---

1) We are indebted to the teachers at the summer school, Martin Kay and Richard Rubinstein, for much good advice during the initial stage of the project. The programming language used is UNIVAC ALGOL, and the program has been run at the regional computer center of the university of Copenhagen (RECKU).

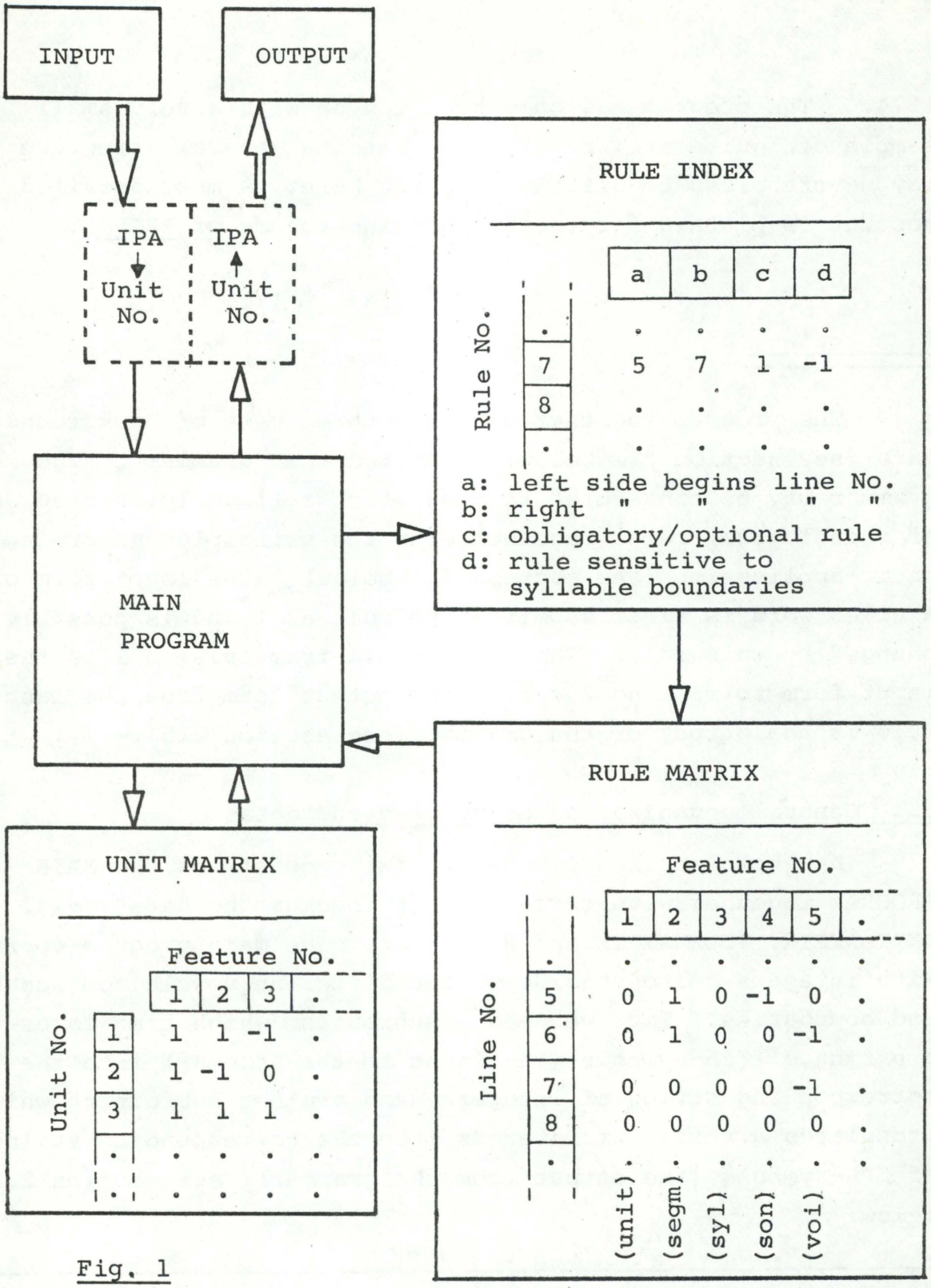
INPUT
OUTPUT
IPA
Unit
No.
IPA
Unit
No.
MAIN
PROGRAM
RULE INDEX
Rule No.
a
b
c
d
7
5
7
1
-1
8
a: left side begins line No.
b: right " " " "
c: obligatory/optional rule
d: rule sensitive to
syllable boundaries
RULE MATRIX
Feature No.
1
2
3
4
5
Line No.
5
0
1
0
-1
0
6
0
1
0
0
-1
7
0
0
0
0
-1
8
0
0
0
0
0
(unit)
(segm)
(syl)
(son)
(voi)
UNIT MATRIX
Feature No.
1
2
3
Unit No.
1
1
1
-1
2
1
-1
0
3
1
1
1

Fig. 1

UNITMATRIX is a two-dimensional integer array, the dimensions being unit-no. (i.e. the integers corresponding to every vowel, consonant, and boundary) and feature-no. (i.e. the integers corresponding to every distinctive feature, like "segment", "syllabic", "sonorant", etc.). Binary features may have the coefficients 1 or -1, a multivalued feature like "height" may have the features 1, 2, 3, etc. The coefficient 0 (zero) is used in cases where the feature is phonetically undefined (e.g. the feature "lateral" is undefined for a boundary, i.e. for a unit which has the coefficient -1 for the feature "segment"). The matrix is thus not redundancy-free, but contains only phonetically meaningful specifications.

RULEMATRIX is a two-dimensional integer array, the dimensions being line-no. and feature-no. To each rule corresponds a number of consecutive lines in RULEMATRIX (e.g. to rule no. 7 correspond lines no. 5-8, see fig. 1). Each line number refers to a unit (i.e. a segment or a boundary in the phonological sense) in the left side or right side of a rule (see section 2.2 below). Only the feature coefficients which are crucial for the correct application of a rule, i.e. (by and large) the features which are mentioned in the standard notation of a phonological rule (see below), are specified in RULEMATRIX, all other coefficients are 0 (zero).

RULEINDEX is a two-dimensional integer array with rule-no. as one of its dimensions. The first number of the other dimension indicates the line number (in RULEMATRIX) where the left side of the rule begins, the second number indicates the line number where its right side begins, the third number indicates whether the rule is obligatory or optional, and the fourth and (for the moment) last number indicates whether the rule is sensitive to syllable boundaries or not. Thus RULEINDEX contains two types of information: it governs the data of RULEMATRIX

(this has the consequence that the feature specifications of a rule can be stored everywhere in RULEMATRIX), and it contains all information which considers the rule as a whole, i.e. its domain and other general conditions for its application.

2.2 The format of phonological rules

(i) is the traditional notation of a phonological rewrite-rule which devoices obstruents before voiceless segments:

(i) [-son] $\longrightarrow$ [-voi] / ____ [-voi]

(i) can be rendered in the format of a phonological transformation instead, viz. as (ii):

(ii) [-son] [-voi] $\Longrightarrow$ [-voi] 2
1 2 1

(i.e. if an input string consists of a [-son]-segment (1) followed by a [-voi]-segment (2), then the non-sonorant, i.e. obstruent (viz. 1), gets the specification [-voi], while the voiceless segment (viz. 2) remains unchanged). This is the format we use for storing phonological rules in RULEMATRIX.

Each unit of the rule (i.e. segment or boundary in the phonological sense) is represented by a line in RULEMATRIX, and we impose the restriction that there must be the same number of lines (in RULEMATRIX) corresponding to the two sides of a rule. This restriction does not mean, however, that we cannot handle deletion, since we use the feature "unit" as a common denominator for segments ([+unit, +segment]) and boundaries ([+unit, -segment]), in agreement with a suggestion of SPE (p. 359, footnote 14). Thus [-unit] means a blank (Ø in the rule algebra), which in UNITMATRIX is defined as -1 "unit", all other features being unspecified, i.e. zero. A deletion rule is a rule whereby a unit of the left side changes into a blank.

Let us now look closer at rule (ii), and let it be rule no. 7 in the ordering. It is stored in RULEMATRIX in four consecutive lines (i.e. lines no. 5-8 in fig. 1). The two latter lines, which correspond to the right side of (ii), only contain the specification that segment no. 1 of the rule gets the coefficient -1 for the feature "voiced", all other coefficients are unspecified (i.e. zero). The two first lines contain the specifications -1 "sonorant" and -1 "voiced", of course, but also +1 "segment" for both units (otherwise a boundary (which is unspecified as to the features "sonorant" and "voiced") in the input form to the rule would, incorrectly, be compatible with any of the segments of the left side of the rule (ii); any blank in the input form to a rule must be ignored, see the following section).

2.3 The application of an obligatory rule to a form

Let us see what happens when we apply rule (ii), which is rule no. 7 in the ordering, to a form. The output form from rule no. 6 is a string of integers where each integer is a unit-no. First of all, RULEINDEX is consulted to see whether the rule is optional or obligatory. If it were optional, we would go to the next rule in the ordering (see further section 2.4 below), but let us say that it is obligatory. RULEINDEX also indicates whether the rule in question is sensitive to the occurrence of syllable boundaries. If this is not the case, the syllable boundaries (which have been inserted by earlier rules) of the input form to the rule are ignored when the program determines whether the rule can be applied (see below). The program must also ignore any blank which the input form to a rule may contain (resulting from the application of earlier deletion rules); otherwise deletion rules could never create new environments for later rules to apply in.

Furthermore, RULEINDEX indicates at which line-no. the left and the right side of the rule begins, and because of the restriction that the two sides of the rule must be stored in the same number of lines in RULEMATRIX, and because all lines of a rule are adjacent, we know the position of each unit of the rule in RULEMATRIX. The program now examines whether any substring of the input form is compatible with the structural description (i.e., the left side) of the rule. This comparison is carried out between two ordered sets of feature numbers at a time, viz. line-no. (in RULEMATRIX) and unit-no. (in UNITMATRIX). There is compatibility between a line of the left side of the rule and a given unit in the input form of the rule if it is true for all features either that the coefficients of the feature are the same, or that one of them is zero. (It is this function of zero which guarantees that zero can never be used improperly as a third value of a binary feature, or a $n+1^{th}$ value of a n-ary feature.) If there is compatibility between a substring of the input form to the rule and all lines of the left side of the rule, then there is "full compatibility".

If no substring of the input form has full compatibility with respect to the structural description of the rule, then rule no. 7 is quitted and we turn to rule no. 8. But if there is full compatibility, then the units of the substring in question of the input form get the feature coefficients which are specified in the right side of the rule. All feature coefficients which are unspecified in the right side of the rule are kept unchanged in the input form. The program then consults UNITMATRIX in order to find the unit-no. corresponding to each of the units which have been changed by the rule (we impose the restriction that all derived units must be defined in UNITMATRIX). An ordered set of feature coefficients is defined as a certain unit-no. in UNITMATRIX if it is true for any feature either that the coefficients of the feature are identical or

that one of them is zero. (Thus a unit which has been deleted, i.e. which has got the specification [-unit] by the rule, will be defined as zero (a blank) in UNITMATRIX since blank in the matrix is unspecified for all other features than [-unit].) The output form of the rule is thus a string of unit numbers, and this output form is then the input form to the following rule in the ordering, i.e. rule no. 8.

2.4 Optional rules

That a rule is optional means that it need not be applied even though its structural description is satisfied. Two alternative ways of handling optional rules within a program like ours present themselves: either we could indicate some stylistic value together with the input form to the grammar, and then specify the stylistic conditions for the application of each optional rule; or we could follow all possible paths through the derivation. We have chosen this latter procedure since it cannot destroy useful information, in contra-distinction to the former alternative (see below).

We adhere to the following strong working hypothesis on optional rules: all optional phonological rules are stylistic, i.e., the non-vacuous application (see below) and the non-application of an optional rule give output forms which differ as to level of style (formality, or the like), and, furthermore, the non-vacuous application of an optional phonological rule gives an output form which belongs to a "lower" (less formal, etc.) level of style than the non-application.

Each time a form is input to an optional rule, the rule is first skipped, and the form is input to the following rule (if this is also an optional rule, then this rule is skipped too, etc.). When such a path of derivation has been followed through, then the program returns to the last optional rule and this time tries to apply it, and so forth. If the grammar

contains n optional rules, each input form will thus follow 2^n different paths of derivation. However, most of these will only differ in uninteresting ways (e.g. when the distinction is one between skipping an optional rule or trying to apply it in cases where its structural description is not met, i.e. is not satisfied for any substring).

The interesting case occurs when the structural description of an optional rule is met and its application will be non-vacuous (i.e. when the input form and the output form of the rule are distinct, i.e. do not consist of the same string of unit numbers). In such cases the program adds the designation L (for "lavsprog", i.e. "low style") to the output form from the rule if it is applied, and H (for "højsprog", i.e. "high style") if it is not applied. The printout contains (among other things, see below) the information L or H concerning the application or non-application of each optional rule whose application to the input form in question would be non-vacuous.

This stylistic information concerning the output opens up exciting perspectives, in addition to the fact that it could falsify our hypothesis on optional rules: Are some or all output forms with both L and H in their derivational history impossible since they are stylistically "incompatible" (e.g. is a form like [le:y̥] of leve impossible since shwa-assimilation is low style, and the keeping of the postvocalic [v] (instead of [u̯]) is high style, compare the normal high style and low style pronunciations [le:və, le(:)o], where [o] in [le(:)o] is derived via [u̯ə])? Can the optional rules be arranged into a "stylistic hierarchy" so that the L-application of a certain rule excludes the H-(non-)application of another rule but not inversely? Does a greater number of L-applications correspond to a more markedly low style pronunciation (as judged by native speakers of the language)? And so on.

2.5 Input and output

The dotted box in fig. 1 indicates the subroutines which translate from IPA-notation (in the input to the grammar) into unit-no., and from unit-no. into IPA-notation. Thus every printout will be in IPA-notation.

The printout corresponding to a given input form to the grammar will consist of this input form together with all the different output forms, and selected information on the derivational history of any path through the grammar. This derivational history consists of all intermediate forms which differ in interesting ways (see the preceding section), together with a letter for each rule in relation to each form: L, H (on these two letters, see the preceding section), O (meaning "the rule has not been skipped, but there was not full compatibility"), V (meaning "vacuous application", i.e. "there was full compatibility, but the input form equals the output form"), A (meaning "non-vacuous application of an obligatory rule", i.e. "there was full compatibility, and the input form was distinct from the output form"; A for obligatory rules thus corresponds to L for optional rules).

3. Concluding remarks

The preliminary nature of this report has already been emphasized. Our first task will be to fill a lot of relevant data into UNITMATRIX, RULEMATRIX and RULEINDEX, and to test the grammar with as many and as varied input data as possible.

This program ought to be coordinated with Peter Holtse's work on speech synthesis by rule of Standard Danish (see his report in the present volume of ARIPUC). The output forms of our program should be used as input forms to the rule synthesis

program, and at present we (i.e. PH, HB and KK) are trying to make this possible. The perspective of this cooperation seems very interesting to us: we may then get an "external test" of the relevance of our phonetic representations, and the borderline between the two programs may turn out to give substance to a distinction between "phonological rules" (contained within the program reported here) and purely "phonetic rules" (contained within the rule synthesis program).