Complementation introduced in linguistic re-write rules

H.C. van Leeuwen*

Abstract

For the purpose of grapheme-to-phoneme conversion in text-to-speech systems, in many applications rule based systems are used. The rule format used is often taken from the area of linguistics. In these so-called re-write rules only concatenation and alternative operators are usually provided for. With these the frequently occurring exceptions in pronunciation cannot be handled elegantly and insightfully. If a complement operator were also available, this drawback would be overcome. The introduction of a complement operator, however, gives rise to some serious interpretation problems. This article describes these problems, and offers a solution to them in the form of an insightful and clear definition.

Introduction

In text-to-speech systems, grapheme-to-phoneme conversion is one of the essential processes [1,2,5,8]. Many systems [4,6,7,9,10] derive their knowledge of how to convert graphemes (letters) into phonemes from a set of linguistic rules, rather than from a dictionary-lookup table.

Once one has decided to use rules to formulate linguistic knowledge, one needs to formalize the syntax of the rules and what exactly they mean (semantics). From the area of linguistics, a widely used formalism is available in Chomsky and Halle [3]. For the application of grapheme-to-phoneme conversion, one can interpret it as a set of context-sensitive re-write rules. The general rule format is as follows:

\[ F \rightarrow C / L \_ R \]  \hspace{1cm} (1)

and should be interpreted as: a certain focus \( F \) in the input stream must be transcribed into the structural change \( C \) if \( F \) is preceded by a left context \( L \) and followed by a right context \( R \). \( F, L \) and \( R \) can be considered as contexts, i.e. structures which should be present in the input stream. \( C \) represents a structure which should be added to the output stream. An example of how these rules can be used is the following simplified rule for the pronunciation of the grapheme ‘c’ in English:

\[ c \rightarrow /s/ \_ \{ e \} \hspace{1cm} (2) \]

This rule states that the grapheme ‘c’ must be pronounced as an /s/ if the ‘c’ is followed by either an ‘e’ or an ‘i’ in the input text. No restrictions are imposed on the left context. This rule provides the correct pronunciation for the letter ‘c’ in words like ‘acid’ and ‘ceiling’. In rule (2) the braces denote the alternative operator: of all structures listed on top of each other (in this case the letter ‘e’ and the letter ‘i’), at least one must be present in the correct position in the input text (in this case immediately following the focus ‘c’).

To compose contexts such as \( F, L \), and \( R \), the linguist has at his disposal primitives and operators. The primitives of the formalism refer to exactly one character in the input (grapheme) or output (phoneme) stream. One manner to do this is by direct reference. In (2), c, e and i refer directly to graphemes in the input stream and /s/ refers to a phoneme in the output stream. Another way to refer to individual characters in the input or output stream is by using linguistic features. In linguistics, features are associated only with phonemes. In the formalism presented here, however, features may also be associated with graphemes. The idea is that one can refer to a set of characters using features, one of which is required in (for instance) the input stream. The entities which can be referred to (the graphemes and phonemes) make up a finite set.

*Institute for Perception Research, PO Box 513, 5600 MB Eindhoven, The Netherlands
The second syntactic element of the formalism, are the operators. The combination of operators and primitives provides the linguist with a tool to compose any context he wants. Two operators are essential to provide him with the power of expression he needs. These are the concatenation operator ‘,’ and the alternative operator ‘{ }’.

The concatenation operator is used to denote a sequence of characters. For instance, specifying that the string 'out' must be present to the right side of the focus is done as follows:

\[
F \rightarrow C \_ \_ o, u, t
\]  

The alternative operator has been demonstrated in (2). It can have any number of arguments.

Given the fact that set of primitives is finite, any context can be specified with these tools. For all positions, separated by the concatenation operator ‘,’ any set of characters, one of which is desired in the input stream, can be specified by the alternative operator ‘{ }’ by means of enumeration. However, concise and insightful rules are cannot always be formulated. If, for instance, a linguist wants to generalize over all consonants except the ‘c’, he could enumerate them, or do some smart manipulation with features. A close study of such a structure would be needed to determine what exactly was meant. The only way to do this insightfully is by using two extra operators, thus expressing exactly what he wants:

\[
[\text{cons} \neg c]
\]

The braces, ‘[ ]’, represent the simultaneous operator: all arguments (in this case ‘cons’ and ‘\neg c’) must be present. ‘\neg’ represents the complement operator: ‘\neg c’ means something like ‘not c’. Context (4) therefore means “all consonants except ‘c’”. These two extra operators provide the linguist with a tool to handle exceptions elegantly. And because of the frequent occurrence of exceptions in natural language, it is desirable to include these operators in the formalism.

The formalism as proposed by Chomsky and Halle does not provide these two operators. Therefore, special attention must be given to their introduction. The simultaneous operator does not lead to interpretation problems: given a certain position to start matching the structure to the input stream, all specified arguments can simply be matched. The structure matches if and only if all its arguments match.

The complementation operator, however, does give rise to interpretation problems.

The Complementation Operator

For simple structures, as in (4), the interpretation seems straightforward, although a formal definition of complementation has not yet been given. If the rule syntax does not put any restrictions on the combination of operators, the interpretation becomes less clear as the structures get more complicated, and may even become ambiguous. Consider the following right context:

\[
\neg \{ a \_ o, u \}, t
\]

The intuitive interpretation of this structure is something like “I want a ‘t’ somewhere, which should not be preceded by a single ‘a’ or the sequence ‘ou’”. There is a vagueness about this interpretation, for example: “What is the exact position of the ‘t’, and what if ‘t’ is not preceded by ‘a’, but by ‘aa’?”

These problems can be eliminated by imposing restrictions on the syntax of the formalism. To be precise, such ambiguities would be solved if complementation were restricted in use to structures with a fixed length only, which would mark (5) as illegitimate.

However, linguists, though not aware of formal problems such as the lengths of the different alternatives not being equal, are able to give an interpretation. The string ‘at’ is felt to be incorrect, because of the presence of the complement operator. For the same reason, ‘out’ is incorrect. The string ‘bt’, on the other hand, is felt to be correct, since the ‘b’ does not match the a, and the ‘t’ does match the t; these are exactly the kind of matching values we want. In the same way, ‘bet’ is correct, this time because ‘be’ does not match ou, and again, the succeeding ‘t’ matches t. ‘aat’ will also be considered correct, for the same reasons as ‘bet’. This is perhaps surprising because the first ‘a’ matches a (which we don’t want), and the second ‘a’ does not match t (which we do want), thus
providing two reasons to reject the string. All the same, the first way to divide the string, in 'aa' and 't', apparently has more weight, and the string is accepted. The string 'att' is very curious in this sense. Dividing it into 'at' and 't' would lead to rejection along the same lines as 'art'. Linguists judge 'att' incorrect. Finally, a string like 'sport' is incorrect, although neither an 'a' or 'ou' is found in front of the 't'. The 't' is not present in the desired position, which apparently must be the second or third position.

These linguistic 'intuitive' judgements can be done justice in formal descriptions.

The first point to put forward is that complementation must be defined with regard to a universe. And apparently, this universe is defined by the structure being complemented itself. In the complemented structure in (4), '–c', the universe consists of the set of all graphemes, '–c' therefore meaning "all graphemes except 'c'". In (5), a structure is being complemented which contains a string of length 1, a, and a string of length 2, ou. The universe for this complementation therefore is the set of all strings whose lengths are 1 or 2. No strings of length 4 are included, and therefore the string 'sport' is incorrect.

A second point to put forward is the notion of 'explicit nofit'. An explicit nofit occurs if there is a possibility to divide an input string in such a way that both the complemented structure (O) and the succeeding structure (S) match at the same time. In the above examples, this was possible for the strings 'at', 'out' and 'att'. In general, for (5) all strings beginning with 'at' or with 'ou' have this property, and will be considered 'explicit nofits'. And apparently, for linguists this is a very strong notion; although one can also divide 'att' in a different, correct, way, the string is judged to be incorrect nonetheless, as begins with 'at', which is an explicit nofit.

The third point is that a so called 'boolean interpretation', an interpretation which comes to mind very easily, does not hold for structures like (5). This boolean interpretation draws an analogy with one of the laws of de Morgan in boolean logics:

\[-(a \lor b) \equiv (\neg a \land \neg b)\]  \hspace{1cm} (6)

In the same way, in this rule format, one could specify:

\[-\left\{ \begin{array}{l} a \\ o \end{array} \right\} \equiv \left[ \begin{array}{l} \neg a \\ \neg o \end{array} \right]\]  \hspace{1cm} (7)

The alternative operator '{ }' resembles the boolean 'or'-operator '
\lor', and the simultaneous operator '[]' resembles the boolean 'and'-operator '
\land'. In (7), the boolean interpretation holds. Complementing an "'a' or an 'o'" means: at the same time we do not want an 'a' and we do not want an 'o', any other character in the input text satisfies the context.

In (7), the complemented structure contains strings of equal lengths (in this case 1). If one specifies contexts like (5), where the structure being complemented contains strings of different lengths, one could ask if the laws of de Morgan still hold:

\[-\left\{ \begin{array}{l} a \\ o,u \end{array} \right\},t \equiv \left[ \begin{array}{l} \neg a \\ \neg (o,u) \end{array} \right],t\]  \hspace{1cm} (8)

In (8), the right hand side can be simplified. There are two requirements to be met by the input. The first is \(\neg a,t\): the first character may be any character except 'a', the second character must be a 't'. The second requirement is \(\neg (o,u),t\) (here the parentheses define the scope of the complementation): the first two characters should differ from 'ou', and the third must be a 't'. Together, this simplifies to

\[\left[ \begin{array}{l} \neg a \\ \neg (o,u) \end{array} \right],t \equiv \neg a,t,t\]  \hspace{1cm} (9)

Note that \(\neg (o,u)\) is already guaranteed by \(\neg a,t\). And concerning this simplified expression (9) we have the feeling this was not meant by (5).

The problem arises because within the complemented alternative structure, strings of different lengths are found, and so the position of the succeeding structure (in this case t) is not fixed. In the boolean interpretation this results in mapping the succeeding structure on different positions in the input stream, and demanding it be present on both positions.

As can be seen in the examples, linguists do not demand correct matching values for all possible divisions (this would be exactly the boolean interpretation), but only for at least one (3) division.
They add the restriction that no division ($\exists 3$) must be possible which leads to an explicit nofit (the complemented structure ($C$) and the succeeding structure ($S$) present at the same time). In fact, this is a procedural description in words to find the matching value of a structure like (5). More formally, one can define complementation in the formalism presented here as follows. A certain input string will match if and only if:

$$\exists \text{ division (C is absent} \land S \text{ is present}) \land \neg \exists \text{ division (C is present} \land S \text{ is present}) \quad (10)$$

An interesting aspect is that, according to this definition, no judgement is given if there is a division for which $S$ is not present. It is neither a possibly correct solution, nor an explicit nofit. Only if for all possible divisions $S$ is not present will the string be rejected. It is important to notice that if $S$ is not present in a certain division, the possible presence of $C$ is ignored.

Discussion

Although I gave only one example, (5), and structured the linguists' judgement on sample input strings, I claim it is a typical case, and therefore the complement operator is now defined satisfactorily.

Problems arise only when more than one division into $C$ and $S$ is possible, where the different divisions can have different matching values. A division into $C$ and $S$ is only possible when the whole structure has a complemented part $C$ and a succeeding part $S$. Different divisions are possible only if the universe generated by the complemented structure contains strings of different lengths. The only way to get such a universe is to complement a structure which itself contains strings of different lengths. And, finally, the only way to construct such a structure is by using the concatenation operator ‘,’ in combination with the alternative operator ‘{‘}. So in fact, (5) is the simplest structure that gives rise to ambiguities. As $C$ and $S$ in (10) denote general structures, any more complicated structure can be simplified recursively to $\neg C, S$, for which we now have a clear and simple interpretation.

References


