Pheatures Speadsheet:
How to Customize the Features

Floris van Vugt
2010

1. Orientation: Customizing the feature system

The feature system employed by Pheatures Spreadsheet is entirely customizable. You can change the set of base symbols, their feature values, the layout of the symbols in the chart, and the diacritics. This is not an easy task, for reasons that will be clear below, and you should take it on only if you have serious reasons to change the features.

There is no user interface for editing the feature system, but it can be done using a spreadsheet program such as Microsoft Excel or OpenOffice Spreadsheet.

2. A few assumptions about the feature system

First we will discuss briefly how the feature system works. It is defined by:

- **Base Symbols** — the base symbols as they have been introduced in the main manual. They have a label that is used for displaying them (e.g. [a] or [p]) and a number of feature settings, that is, a feature matrix. This is stored in the file **basefeatures.txt**, whose syntax will be discussed in 3.3. It is important to note also that the features that occur in basefeatures.txt will be taken as an exhaustive list of all features in the feature system. That is, you cannot make mention in another file of a feature that does not also occur in the base symbol list.

- **Diacritics** — These have a label and a feature matrix with requirements and one with the feature changes they effect. They are defined in the file **diacritics.rules** (3.3). Some more clarifications on how diacritics are used are presented in 2.2.

- **Dependencies** — Dependencies are used in the interface. When certain feature changes are made, other feature changes follow–up automatically. For example, when the user changes sounds to [+high], they are automatically changed to [–low]. Dependencies are defined in **dependencies.rules**.

- **Contradictions** — Certain feature combinations are contradictory, such as [+front, +back], and we want to warn the user about them. These contradictions are defined in **contradictions.rules**.

- **Chart definition** — Finally we want to present all possible sounds in a chart that is similar to the way the IPA alphabet is typically presented. This is defined and can be

customized in three files, **ipachart-consonants.txt**, **ipachart-other.txt** and **ipachart-vowels.txt**.

2.1 Diacritics

A diacritic is attached to a base symbol and effects some feature changes to that symbol. Diacritics may pose requirements on what sounds they can attach to, in the form of a feature matrix that the sound must match.

Here are a few observations about how Pheatures Spreadsheet views diacritic attachment.

2.1.1 Feeding and bleeding and multiple diacritics

Let us take the example of creating a voiceless aspirated nasal [m̥ʰ]. We find in the diacritic list that the voiceless diacritic requires the sound it attaches to be [+sonorant, +voice], from which [m] is nondistinct. So, using the inventory editor, we can drag the diacritic on to the base symbol [m] to create [–spread gl, –constr gl], which matches [m̥], and therefore we can add it and create the desired [m̥ʰ].

What would have happened if we had added them the other way around? We would run into a problem when trying to add the aspiration diacritic to [m], because that diacritic requires the sound to be [–voice], among other things, and [m] is [+voice]. One can say that the voiceless diacritic has *fed* into the application of the aspirated diacritic.

Similarly, the voiceless diacritic *bleeds* application of other diacritics, such as the breathy–voiced diacritic, since that one can only attach to [+voice] sounds.

To summarise:

- Diacritics pose requirements on the compound sound they attach to, not on the base symbol. Therefore we get feeding and bleeding in diacritic attachment.

2.1.2 The transparency assumption in diacritic attachment

Pheatures Spreadsheet makes an additional assumption about diacritic attachment, which is as follows:

- Every diacritic that is attached must be attached transparently, i.e. it cannot change the value for features that previously attached diacritics set.

Let me give an example, which will be artificial precisely because our assumption is that any sensible diacritic system has this transparency property anyway.

Suppose that we are in a simple feature system with three features: a, b and c. Suppose that we have the base symbol B=[+a, –b, –c]. We have a diacritic x that applies to –a sounds and changes them into [+b, +c], so we cannot form Bx (i.e. applying diacritic x to B). But let's

assume that there is also a diacritic y that applies to [–b] sounds and turns them into [–a]. Then we can form By, which will have features [–a, –b, –c]. Let us also assume that there is a diacritic z that applies to [–a] sounds and turns them into [+a]. Can we then add diacritic x to By, to form Byx? Yes: By meets x's requirement [–a] (so y has fed into x's application).

But can we then add diacritic z to Byx to create the symbol Byxz? No, because this violates the transparency condition. The point is that z changes the value [–a] (that was set by the diacritic x) to [+a], thus as it is "overriding" another diacritic. That is what the transparency condition prohibits.

Another way of putting the transparency condition:

- Given a symbol, the union of all the feature changes of the diacritics cannot contain a contradiction.

In our example: the union of the feature changes of the diacritics in Byxz would be [–a, +b, +c, +a] which contains a contradiction.

This has a useful consequence:

- A diacritic can only be applied once meaningfully to a symbol.

The point is that given the transparency condition, we know that the second application of the diacritic would not effect any feature changes.

This means that even though we allow feeding and bleeding in our diacritic system, the diacritics that are applied to a symbol are essentially a set: if there is one ordering of application that is legal, i.e. that has the correct feeding and bleeding relations among them, then any other ordering that is legal as well will create the exact same feature matrix.

What will go wrong if we violate this transparency constraint? The sim-ple answer is: nothing. You will be allowed to create such symbols in the phoneme inventory editor. The complicated answer is: the labeling algorithm (see main manual) will not find that label when it requires it to try out non–transparent diacritic applications.

Notice finally that this transparency condition counts only for diacritics. We do not wish to say anything about whether our phonological rules can be opaque or not!

2.2 Ordering diacritics for display

Since we have now established that the diacritics that apply to a symbol can be seen as a set, we can address the next issue. Technically, the diacritic labels should be added in a particular order so that the unicode symbol will display correctly. That is, we must first add the non–spacing characters (i.e. characters that merely attach above or below the base symbol such as the voiceless diacritic), and only afterwards the spacing characters (such as the aspiration diacritic). If we would do it the other way around, the voiceless diacritic would not appear straight

underneath the base symbol but be displaced to the right, since it takes into account the space taken up by the aspiration h.

How is this ordering of attachment defined? Pheatures Spreadsheet quite simply registers the order in which you give the diacritics in the file **diacritics.rules**, and for any symbol, it will attach the diacritic labels in that order. For example, the voiceless diacritic will appear earlier in the file **diacritics.rules** and that will cause it to always be applied before the aspiration.

Notice also that this ordering of application of diacritic labels is independent of the application of their feature changes! This is important.

3. Input files and their syntax

Now that we have covered the fundamentals of our feature system, we are ready to look into how to customize them by editing the files. Pheatures Spreadsheet might have come to you in a compressed archive, called jar. This is a standard archive method for Java programs and its convenience is that it yields a single file that contains all program requirements as well as data files.

In order to edit the feature system you will first have to extract this archive (a zip file extractor will do the job; these are easily obtained). Extract them in a folder and make sure it recreates the folder structure present in the jar archive.

Then, after extraction you will notice a number of program files (which end in **.class**) and a number of folders. All files configuring the feature system are found in the folder **data/**.

3.1 How to run the program in this extracted form?

You will need to invoke the main class, which is **Pheatures.class**, which is located in the root folder of the jar archive. You can often do this by double– clicking or invoking **java -jar Pheatures** from the command–line. Check the documentation of your Java Runtime Environment for your operating system.

If you want to put the files back in archive form after you edited them, you can simply compress the folder you previously extracted back into a jar archive, which you can then run and send in the same way as the original Pheatures Spreadsheet package you downloaded.

3.2 Finding files and editing

In data/, you will find two types of files that require slightly different ways of editing: files that end in .txt and files that end in .rules, and these will be discussed in the following sections.

Whenever you edit these files, make sure that your editor supports Unicode, since all of the symbol characters as well as diacritics in Pheatures Spreadsheet are entirely Unicode–based.

3.2.1 What are tab–separated files (.txt)?

The .txt files that are used by Pheatures Spreadsheet are not ordinary text files. They are tables, and they are saved in a tab–separated format since this was thought to be compatible with the largest number of existing spreadsheet editors.[1] In these files, the rows of the table are the lines in the file, and the columns in each row are separated by a tab (whitespace) character. Contrary to usual csv files, the values are not enclosed by quotes (**"like this"**) but stored plain (**like this**).

You can open and edit these tab–separated files in a text editor (such as Windows Notepad or GNU Emacs), but it is far more convenient to open them in a spreadsheet program such as Microsoft Excel or GNU Numeric.

Upon opening, you might be asked for a field separator. Make sure you enter only tab as a field separator and not comma or semicolon (;) as well as that might have unexpected results. Furthermore what is often called "field encloser" should be absent, as no quotes surround the cells.

Finally, make sure that you do not select to merge delimiters! In some applications, multiple delimiters are considered as one. But for our purpose, when multiple tabs occur that reflects that the cells they separate are empty. To ensure correct horizontal line–up, we should therefore not merging delim-iters when reading or writing these tab–separated files.

Obviously, these same settings should be used when saving the files.

3.2.2 What are rule files (.rules)?

The .rules files are not tables. They are essentially a list, the exact contents of which vary, and they will be discussed when we discuss each of the files. You can open, edit and save these files from your favorite plain text editor such as Windows Notepad or GEdit. Make sure you do not use a word processor as that may have unexpected results.

3.3 Syntax of the particular files

basefeatures.txt

This file defines the base symbols and their features. The first row is a header, giving the names of the features.

As explained before, this is taken to define an exhaustive list of all feature names. You cannot mention a feature elsewhere that does not appear as a column in this base symbol table. Furthermore, the order of the columns in this file defines the order of the features in the drop–down lists in the selection panel of the main user interface.

[1] Some spreadsheet editors are able to read these files in their csv–mode (Comma– Separated Values), where for these programs the "comma" is really a "tab".

The feature names, by the way, can contain spaces or other characters, as long as they do not contain tabs (since this will cause confusing when these files are parsed by Pheatures Spreadsheet).

Starting from the next row are the base symbols. The first column gives the (Unicode) label of the symbol. The second column contains a reference to a sound file (which may or may not be actually implemented in the version you are using — if you are not sure what to do, leave it empty).

Then from the third column onwards you can enter the values for the feature that that column represents. Possible values are plus (+), min(-)[2] or null (0).[3] Do not leave the field empty but write 0 instead. Null has to be explicitly coded for parsing purposes and it will be clearer for you when you edit the file.

diacritics.rules

In this file each row defines a diacritic. The format is as follows, where the parentheses are not part of the format, but the semicolons (;) and arrows (>) are:

(1) (*description*) ; (*label*) ; (*requirements*) > (*changes*)

The items represent:

- (*description*) This is a (short) name that represents the diacritic. It must be unique, i.e. there cannot be two diacritics with the same description.

- (*label*) This is the actual symbol that should be pasted onto the base symbol when we apply the diacritic. This label can be given in one of two ways. Either it is just literally a Unicode symbol, or, alternatively, its numeric character code. The reason for this alternative is that it may not be straight–forward to enter a diacritic label in your text editor without a base symbol to which it can attach. Therefore it may be easier to enter the numeric character code, which can be found in any Unicode chart.[4]

- (*requirements*) This is a feature matrix (without enclosing square brack-ets) that defines what features the sound must have to which this dia-critic attaches.

- (*changes*) Again a feature matrix, representing what feature changes this diacritic effects.

An example is the following line. Notice that any whitespace in between the items is removed during parsing. This leaves you the freedom to vertically align the items so that they are

[2] Notice that there are multiple Unicode characters corresponding to -(longer and shorter dashes). To avoid errors, it is safest to copy a minus from another cell into where you want it, though in most cases the minus from your keyboard should be the right one.

[3] That is, the number 0. Not the letter o, nor uppercase O. Again, if you are confused, simply copy some of the zeros from another row.

[4] For an excellent resource, see http://www.phon.ucl.ac.uk/home/wells/ipa-unicode.htm.

maximally readable for you. Note also that we have here used the numeric value (805) of the voiceless diacritic, rather than write it as a Unicode character.

(2) `voiceless; 805; +sonorant, +voice > -voice`

Remark also that the order in which you give the diacritics in this file, is the order in which the labels are applied to a base symbol (see 2.2).

`dependencies.rules`

Dependencies are changes that are applied automatically when the user se-lects to change certain features. Again, each line represents one such depen-dency. The format is as follows:

(3) (*conditional*) > (*changes*)

- (*conditional*) A feature matrix (without enclosing square brackets) rep-resenting the features that, when they are changed, initiate this depen-dency.

- (*changes*) A feature matrix representing the feature changes that occur automatically when those in the conditional are changed.

An example is the following, which expresses that when a sound becomes consonantal, it cannot have a tense value anymore. When the user makes this change in the interface, a grey message will appear that this change has been filled in.

(4) `+consonantal > 0tense`

`contradictions.rules`

This file is one of the simplest: each line represents a single feature ma-trix specifying a contradictory combination of features (without surrounding square brackets). For example:

(5) `+consonantal, +tense`

This line specifies that for a sound to be both +consonantal and +tense is contradictory. In the interface it would appear marked in red. When the user attempts to write a rule that changes a sound to [+consonantal, +tense] a red warning message will appear.

`ipachart-*.txt`

Perhaps the most tricky files to edit are those that define the IPA–like chart. They are used by the program to lay out the base symbols in the inventory editor, as well as in the Chart view in the main screen. It is crucial to edit this file if you add or remove base symbols, since all base symbols must be present in this chart, and any symbol that appears must be a base symbol.

You can open the `ipachart-X.txt` in your spreadsheet editor (where X is either `consonants`, `other` or `vowels`). Each cell is one of the following:

- **Empty**. When you leave no character in the cell, it will be treated as empty in the chart.[5]

- **A label**. You can enter text that will appear in the phoneme inventory editor to clarify what each row or column represents; or you could write the comments in the middle of the table if you so prefer. Simply enter your text surrounded by square brackets ([like this]). Any spaces you write in this label will be converted into line breaks when the cell is displayed. Try it out. This will make it easier to fit all the text you want in the relatively small cells. If you do not want such a line break, simply remove the space or substitute it by a dash ([`like-this`]) will be easily readable as well.

  Notice that labels are not shown in the Chart view in the main screen. This is thought to be too confusing. They are only shown in the phoneme inventory editor.

- **A symbol**. That is, a base symbol, or a base symbol with some dia-critics added. For one, all base symbols should have a place in one of the three charts (that is, in the consonant, vowel or other chart). If the program encounters a base symbol that does not have a place in either of these charts it will give an error message on the command–line. You should fix this before you continue to use the program, otherwise unex-pected things will happen. For example, the symbol will not be shown in the Chart view.

  In addition to the base symbols you may add some base symbols with added diacrtiics. This will make them easier available to the user when they build the phoneme inventory, because then he or she does not have to drag it onto the base symbol him– or herself.

  What is the format for entering a symbol? If you want to add a bare symbol, you simply write its label as it appears in basefeatures.txt.

  Make sure you write it in exactly the same way! Unicode symbols that consist of multiple parts can often be written in more than one way, by adding the parts in different orders. It is crucial that it appears in the chart as it does in basefeatures.txt or else it will not be recognised. Again, to be sure, you are advised to copy and paste the symbols from basefeatures.txt.

  How to write a symbol with diacritics? You write the base symbol as above, followed by a semi–colon (;) and then write the names of the diacritics separated by semicolons. So do not write the diacritic labels here, nor paste the diacritics onto the base symbol yourself! This is again for parsing reasons and readability, as well as to avoid confusion. So for example our famous voiceless aspirated nasal [mh] will be written as m;voiceless;aspirated.[6]

[5] As mentioned earlier, make sure that your spreadsheet editor does not merge adjacent delimiters, since then it will "delete" empty cells when it saves and shift all following cells leftward, causing misalignment.

[6] Since, as we explained before, diacritics are essentially a set, you could also write m;aspirated;voiceless to get the same result. The order of the diacritics does not reflect the order of attachment. In fact, you should be careful not to write diacritics that cannot in any order attach to the base symbol! The program does not check this for you.

3.4 Checking

How do you know if your changes do not cause internal errors in the feature system? Pheatures Spreadsheet performs some checking when it starts up. For example, it will check whether it recognised any symbol that you entered into the ipachart-X.txt files. Also, if there is some base symbol that is not there, it will send a warning.

These warnings are written to the standard (error) output. If you run the program in Windows and execute it by double–clicking on it, you will not see these messages. Rather, you need to execute the program from the command–line. Similarly, in Mac OS, you can go to the Console (typically found somewhere in Applications or Utilities).

Make sure you keep an eye on such messages and also keep a copy of any files you are editing so that you can revert whenever you make a crucial or mysterious mistake.

3.5 Closing remarks

With this customizability, it is hoped that linguists can experiment with their own feature systems and investigate how the phonological rules that they propose would work in detail and check them for doing exactly what we want them to do.

It is finally hoped that the program will make learning phonology fun and take out the frustrating aspects of never knowing what exact features sounds are supposed to have.