Pheatures (working title)

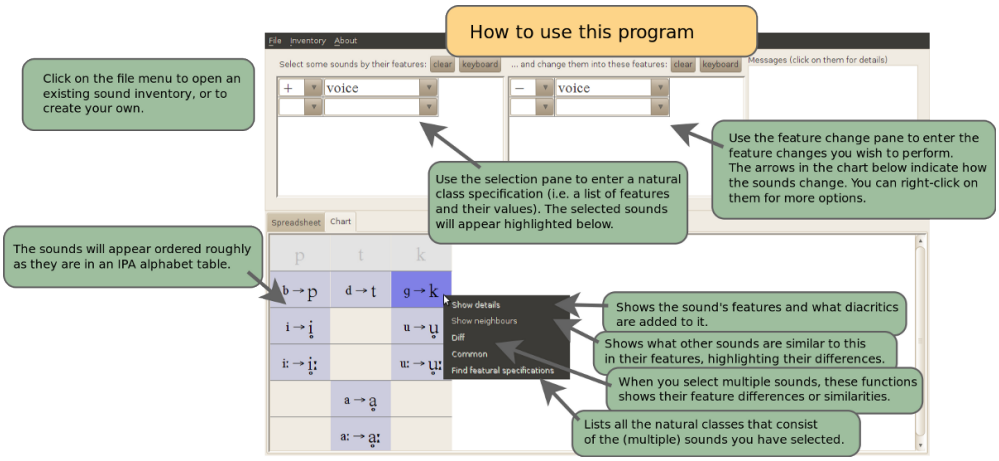
Floris van Vugt

July 2, 2010

Overview

FeaturePad is a program for linguistics students to learn what features the world's sounds have. This is achieved through active practice. The program never tells you the answer to a question, but it checks it for accuracy, and often points out problems with the answer that need to be fixed.

For linguists, FeaturePad provides an interface to investigate properties of the feature system and user–definable phoneme inventories.



This manual will explain how to use the interface. It will also provide comprehensive instructions on how the advanced user can customise the feature system.

Contents

Chapter 1

Using the interface

1.1 What things mean

It would be good to know what a few things are called in this manual and in the program.

1.1.1 Features and feature matrices

For FeaturePad, a sound is really just a list of features and their values, such as: [–consonantal, +syllabic, +low, –high]. Such a list is also often called a *feature matrix*. Sometimes the word *feature* is used more specifically to refer to a feature and its corresponding value in a feature matrix (e.g. one would sometimes say that +low is a feature of the low vowel [a]).

Some of these feature matrices have a symbol as one would find in the IPA alphabet. Examples of such symbols are [a], [ə], [ɓ], [ʃ], [t^h] etc. The latter symbol is interesting because it consists of a base symbol (t) and what we call a *diacritic* (^{h}). So in general we can say that any symbol consists of one base symbol and zero or more diacritics added to it.

Base symbols have their associated feature matrices and every diacritic that we add to them changes a number of features. For example, aspiration (represented by the diacritic ^{h}) will change the sound's feature "spread glottis" to + and the feature "constricted glottis" to –. Another way of saying this is that it causes the feature change [+spread gl, –constr gl].

1.1.2 Phoneme inventory

In FeaturePad we use a featuresystem that contains about two–hundred base symbols, and eighteen diacritics. Since some diacritics put restrictions on what sounds they attach to, not all possible combinations of base symbols with any set of diacritics (200×2^{18}) can be made, but there are roughly $43,000$ symbols that can be created in this way.

Of course, no language uses all of them! That is why we can define a *phoneme inventory*, which is really just a selection of all possible symbols.

1.1.3 Feature matching

Typically phonological rules apply on a subset of the phoneme inventory of a language. For example, a language may have a fronting rule that causes vowels to become [+front], but it does not apply to sounds that are not vowels. In this case it is possible to say that the rule applies to sounds only if they have the feature +syllabic. A technical way of saying the same thing is that the sounds it applies to must be *nondistinct* from the feature matrix [+syllabic], which is to say that for any two features they must have the same value. A computer scientist would say that these sounds *match* that feature matrix.

So if we have a phoneme inventory, then given any feature matrix we can find the sounds that are nondistinct from it. For example, given the English phoneme inventory and the feature matrix [–delayed release, –voice] we would find that the sounds [p], [t] and [k] are nondistinct from it, and only they. That is to say, [p,t,k] is a *natural class* and it is *specified* by the features [–delayed release, –voice]. Notice that in other inventories one of these sounds might be missing, or there may be more sounds that are [–delayed release, –voice].

Then a phonological rule is defined by a feature matrix that tells us what sounds it applies to (so it applies to some natural class) and another feature matrix that tells us what features in those sounds the rule changes.

Now FeaturePad lets you define such a rule and see in real–time what it does.

1.2 The main screen

1.2.1 Get a phoneme inventory

When you open the program, you will not see much yet. That is because we need to start out by opening an inventory or creating a new one. If you click the menu `File` and then select `New phoneme inventory`, you will go to the inventory editor screen (see 1.3). if you select `Open phoneme inventory` you can choose an inventory file that you have previously saved or that you downloaded. In both cases you will see that the list in front of you now shows the sounds in the inventory you selected.

There are two ways of viewing the inventory sounds, and you can switch between them at any time. By clicking `Spreadsheet`, or `Chart` in the middle–left of the screen.

1.2.2 Spreadsheet view

The spreadsheet view will serve to show you the features of each of the sounds that you have selected. The sounds appear in a list and the columns represent the values the sound has for each of the features.

You can restrict the features in this list by entering some feature–values in the selection panel (see 1.2.4) and you will see that the list is updated to contain only the sounds that have these features.

If you have entered a rule, i.e. if you have specified that you want some features to change value, then this spreadsheet will show the sound features *after* the change has applied. You will then also see that there are now two symbols in the leftmost column, separated by a rightward pointing arrow. This shows the original symbol and the symbol after the change. If any could be found, that is (see 1.2.7), otherwise you will see a question mark.

1.2.3 Chart view

The chart view shows where the sounds in the inventory would end up in an IPA chart. That is, the consonants are given on top and the vowels below. The sounds of these two are then ordered roughly by place of articulation (columns) and manner (rows).

As in the spreadsheet view, you can use the selection panel to restrict to a particular natural class of sounds, but contrary to the spreadsheet view,

Figure 1.1: Entering a feature–matrix in the selection panel

the symbols that are distinct from your selection will not disappear but be greyed out.

Again, if you have entered a rule, the symbol representing the changed feature matrix will appear to the right of the arrow.

1.2.4 Entering a phonological rule

Now we will turn to how you can enter a number of feature–values to select sounds. That is, how to enter the left hand side of a phonological rule.

The interface is shown in figure 1.1[1]. Each row represents a feature–value. You can enter the value in the left dropbox, and then the feature in the right dropbox. If you click on them a list of options will appear. In addition to this, you can click on the `clear` button to remove all your feature–value settings, or `keyboard` to enter the feature–values using the keyboard (cf. section 1.2.5).

Now we turn to entering the right hand side of a phonological rule. Specifying these feature–values that you want to change is done in a similar way as you enter the selection.

As you are working with this interface, you will notice that the background of this table occasionally turns yellow or red. This is a signal that

[1]Notice that the precise appearance of the program may change according to your operating system.

something is the matter with the features that you have selected, and this will be discussed in section 1.2.6.

1.2.5 Entering a feature matrix using the keyboard

To enter a feature matrix using the keyboard, you can enter the values in a single line in much the same way you would write them on paper. If you click the button `keyboard` a small new window will appear that contains a text field where you can enter the feature matrix.

You enter the feature–value settings separated by commas and for each the value precedes the feature name. For example:

`-consonantal, +high, +back`

To save time, you could also have written `-conson, +high, +back`, since in the standard feature system there is no other feature that starts with `conson`. Notice that you could not have written `cons` to achieve the same goal, since there is also a feature "constr gl" (constricted glottis).

Entering a natural class with the keyboard can be useful as a faster way to input feature matrices for users that are well familiar with the feature system. Also, you can use this function to quickly enter a rule that you saved in your personal documents or papers by pasting them into the field.

1.2.6 Feedback messages

A third panel on the top right of the main screen is devoted to messages pointing out issues with your feature selections. These messages are shown in a list format and a single click on them will bring up a window that gives more details about the message.

A typical kind of message is signalling *redundancy*. Let us give an example. If you are selecting the low vowels in the inventory, and enter [+low, –high] in the selection panel, a redundancy message will appear. This is because any vowel that is [+low] is automatically [–high]. So you could have selected the same sounds by just choosing [+low]. Notice that the message will not tell you *which* feature is redundant, since it is more instructive for you to try out removing features and discover if any more sounds appear.

Another kind of message tells you about *contradiction*. This would happen for instance if you accidentally specified [+labial, –labial] as feature changes. That is called an *overt contradiction*. But the program is also on the lookout for *implicit contradictions*. These are contradictions that

arise out of what features mean in our feature system. For example [+front, +back] is such an implicit contradiction, since obviously a sound cannot be both pronounced both in the front and the back of the mouth (at least not in our feature system — but it is up to you to customize the feature system if you disagree, and this is explained in chapter 2).

1.2.7 Feature matrix labeling

When you implement a rule the program takes all sounds that match the left–hand side of your rule and applies the given feature changes. The result is a feature–matrix. Next, the program will attempt to find a corresponding symbol. This is not as easy as it sounds. We not only have to compare our feature matrix with those of the base symbols, but also detect if we can arrive at our feature matrix by applying any set of diacritics. This is called *labelling* or *spelling* a feature matrix.

Sometimes no label will be found. That is because there are many more possible feature matrices than there are symbols. In our feature system with 28 features there are 3^{28} different feature matrices, but only some $43,000$ possible spellings (and some of those are synonyms: symbols that have the same feature matrix).

If this happens for any sound that you have created with a phonological rule, you will see a yellow question mark after the rightward arrow, instead of a symbol.

1.2.8 Comparing features: some tools

But this is where the fun begins. Using the spreadsheet, you will be able to see what the features of this newly created sounds are. If you right–click on it in either the spreadsheet or the chart, you can choose `show neighbours`. This will make a new window appear that contains a list of all base symbols and some combinations of diacritics that come close to spelling your newly created feature matrix. The features that are different are highlighted, so that you can see what additional feature changes you might need to make to yield the symbol you wanted.

Furthermore you can access more functions when you select multiple sounds. This can be done by holding down the control key on your keyboard and clicking on the different sounds in either the spreadsheet or the

chart[2].

Once you have selected various sounds in this way, you can right–click on them[3] to bring up a popup menu. In this menu you can select `diff` to show only the feature differences of these two or more sounds. Similarly, `common` will show the features that are the same for them.

These tools are designed to allow you full insight into how the feature system works and gain familiarity with how sounds relate to each other.

1.3 The phoneme inventory editor

Sooner or later you will want to enter the phoneme inventory that corresponds to the language that you are working on. If you click in the main screen on the menu `File` and then `New phoneme inventory` this will bring up the inventory editor starting with an empty inventory. If, from the main screen, you click on `Inventory` and then `Edit phoneme inventory`, you will be able to edit the phoneme inventory that you have currently loaded.

The phoneme inventory contains a big chart with all base symbols on the left, and a smaller list with diacritics on the left. The base symbol chart will appear in a chart that is similar to the way the IPA alphabet is usually laid out.

To include a sound in your inventory, simply click on it once. You will notice that it takes a blue colour, and this means you have succesfully added it. Clicking on it again will remove it from the inventory, but leave a greyed out cell in the table in case you'll want to change your mind and add it later.

To add diacritics to the symbols, you can click on them and hold your mouse button while you drag them to the left on to the base symbol you want. This is called drag and drop, and again the precise implementation may depend on your operating system.

You will see that often a new row will appear in the table below the base symbol you selected and that the base symbol with its added diacritic appears there. This is done to keep the organisation of the inventory editor intuitive. You can add another diacritic to a symbol that has already a diacritic by

[2]Again, depending your platform, the key might be called differently. Consult your familiar programs to see how multiple selections are handled there and it probably works the same in FeaturePad. That is the convience of Java copying the guest operating system "look and feel".

[3]This may once more depend on your operating system.

dragging it onto the diacritic–marked symbol in the main chart.

In some cases, you will not be able to add a diacritic to a symbol. In this case, it will light up red during the dragging and you will get an explanation message. Some diacritics have requirements on what sounds they can attach to. For example, the nasal diacritic (e.g. õ) can only attach to sounds that are [+sonorant, –nasal]. To gain information on this, you can right–click on the diacritic to bring up a screen with details.

Similarly, you can right–click on symbols in the chart to see how they are composed of a base symbol and diacritics and what their features are.

Using the buttons in the bottom of the screen you can save your phoneme inventory to a file or load one from a file[4].

You can close the screen by pressing either `Use inventory` or `Return without using`. If you want the phoneme inventory you have been working on to appear in the main screen, you can click `Use inventory`. Otherwise, click `Return without using` to abandon your changes.

Notice that even when you loaded an inventory from a file, you can safely use the inventory editor to add a sound or two to it, and then click `Use inventory`. The changes you made will not be saved to a file unless you click `Save to file` in the inventory editor, yet you will be able to use the updated inventory in the main screen.

1.4 Natural class functions

Now especially if you are a linguist already familiar with features, you might find the following tools interesting to gain insight in the phoneme inventory of your language.

These tools may be disabled in your copy of the program if you have a pure pedagogical version, since they may make your homework too easy.

1.4.1 Listing natural classes

Given a language's phoneme inventory, what sets of sounds form a natural class? Some classes are obvious, such as the vowels, since they and only they match [+syllabic]. But what other classes there are? In a typical feature inventory, not just any arbitrary set of sounds will be a natural class.

[4]Note that these files are no longer compatible with the previous Windows implementation of FeaturePad.

By selecting the `Inventory` menu from the main screen and then `List natural classes`, you will see a screen where you have to do is click `Start` to have your machine generate a list of all possible natural classes. The set of sounds is given on the left and the features that select precisely this class, no more, no less, are given on the right.

You can double–click on any such class to automatically enter the features in the selection panel, so that you can customize them and try further combinations.

1.4.2 Finding feature specifications

Suppose you have a few sounds in your inventory in mind and you are curious if they form a natural class, and if so, what feature matrix selects precisely that class. Furthermore, quite possibly there are many different feature matrices that all do just that. Here is how to get a list of these possible feature matrices, which we will call *feature specifications* of the set of sounds.

You can select any sounds in the spreadsheet or chart (in the way described in 1.2.8) and then right–click on them and choose `Find featural specifications`. This will bring up a new window where pressing `Start` launches the search for feature specifications of that set of sounds.

If any exist, that is, for quite possibly the set of sounds you have selected simply isn't definable by a feature matrix. That is, for any feature matrix that you give there will either be a couple of other sounds that match it too, or a couple of sounds that are in your set be excluded.

Finally, when the search is over the feature specifications that are not *minimal* will be marked in yellow. A feature matrix is minimal if it succesfully selects the group of sounds you gave, and if there is no other feature matrix that does the same thing but with less feature–value pairs.

If you are only interested in these minimal feature specifications, you can tick the checkbox that says to search only for minimal specifications. The program will then find *all* minimal feature specifications, but not necessarily all non–minimal feature specifications. However, it will be faster, since many search branches can be trimmed as smaller and smaller feature specifications are found.

Chapter 2

Customizing the feature system

The attractive thing about FeaturePad that may make it useful for established linguists as well is that the feature system is entirely customizable. That is to say, all the base symbols can be changed, their features changed, and how they are laid out in the chart; diacritics can be added or removed; one can even add or remove features altogether; and one can edit what contradictions are.

The only caveat is that there is no user interface for editing the feature system. That means it will be slightly technical. However, it does not require programming knowledge, nor sophisticated programs beyond a simple spreadsheet editor such as Microsoft Excel or OpenOffice Spreadsheet.

2.1 A few assumptions about the feature system

First I will discuss briefly how the feature system works, so that the reader will understand what to do in order to customize the desired parts.

The feature system is defined by:

- **Base Symbols** — the base symbols as they have been introduced in 1.1.1. They have a label that is used for displaying them (e.g. [a] or [ɯɻ]) 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 2.2.3. 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** — which have been discussed before. They 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` (see 2.2.3). Some more clarifications on how diacritics are used are presented in 2.1.1.

- **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` (see (2)).

- **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` (see (4)).

- **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`, and they will be discussed in (5).

2.1.1 Diacritics

In section 1.1.1 it has been explained that 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.

A few observations about how FeaturePad views diacritic attachment.

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 [m̥]. Now the aspiration diacritic requires [+consonantal, –voice, –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.

Transparency assumption in diacritic attachment

FeaturePad 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 simple answer is: nothing. You will be allowed to create such symbols in the phoneme inventory editor. The complicated answer is: the labeling algorithm (1.2.7) 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!

2.1.2 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? FeaturePad quite simply registers the order in which you give the diacritics in the file `diacritics.rules` (see 2.2.3), 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.

2.2 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. Standard, FeaturePad 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. Any archive program should be able to extract them. 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/`.

2.2.1 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 FeaturePad package you downloaded.

2.2.2 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 FeaturePad are entirely Unicode–based.

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

The `.txt` files that are used by FeaturePad 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 your enter only tab as a field separator and not comma or semi–colon (`;`) 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 delimiters when reading or writing these tab–separated files.

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

[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".

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 favourite plain text editor such as Windows Notepad or GEdit. Make sure you do not use a word processor as that may have unexpected results.

2.2.3 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.

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 FeaturePad).

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.

[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.

`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 brackets) that defines what features the sound must have to which this diacritic 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 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 2.1.2).

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

`dependencies.rules`

Dependencies are changes that are applied automatically when the user selects to change certain features. Again, each line represents one such dependency. The format is as follows:

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

- (*conditional*) A feature matrix (without enclosing square brackets) representing the features that, when they are changed, initiate this dependency.
- (*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 matrix 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 diacritics 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 unexpected 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`.

[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.

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 [m̥ʰ] will be written as `m;voiceless;aspirated`[6].

2.2.4 Checking

How do you know if your changes do not cause internal errors in the feature system? FeaturePad 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.

[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.

2.3 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.

Chapter 3

Credits

3.1 People

FeaturePad was funded by two grants from UCLA's *Office of Instructional Development* to Bruce Hayes. See their website[1] for more information.

The idea for this program originates with Bruce Hayes[2], who is a professor at the Linguistics Department of the University of California, Los Angeles. He also designed the feature system that comes standard with FeaturePad.

A first implementation in Windows of this idea was made by Kie Zuraw[3], who is also a professor at the UCLA Linguistics Department. She also designed much of the precise implementation of the feature system. The current version of FeaturePad shamelessly copies many of her elegant algorithms.

The current Java–based implementation is designed and programmed by Floris van Vugt[4], a graduate student of linguistics at UCLA. Some additional functionality such as Diff, Common and the natural class functions have been designed (in collaboration with Bruce Hayes) and written by Floris.

[1] **http://www.oid.ucla.edu/**
[2] **http://www.linguistics.ucla.edu/people/hayes/**
[3] **http://www.linguistics.ucla.edu/people/zuraw/**
[4] **http://florisvanvugt.free.fr/**

3.2 Materials

The program uses the Doulos SIL Unicode–based IPA font, which is made available online[5] by SIL.

The program is compiled using Sun's Java Development Kit, version 1.6. It requires JRE 1.5 or higher to run. I recommend using Sun Java JRE, although other implementations have been tested as well. Mac OS X's java implementation has issues with the drag and drop interface, which may make the Inventory Editor unusable. Other than this the program should work fine on all platforms.

[5] **http://www.sil.org/**