Asheninka User Documentation

H. Andrew Black
SIL International
asheninka_support@sil.org
21 November 2025
Version: 1.2.0

Contents

1 Introduction

The **Asheninka** program is a linguistic tool with two goals:

1. Explore various syllabification algorithms.
2. Provide a principled way to insert discretionary hyphens in a list of words (which can then be used for typesetting text).

The first is clearly a linguistic research goal. The second is a practical application of the first goal. While one probably will want to use IPA (IPA 2016) for the first, the second should be done using the practical orthography.

2 General user interface items

This section notes some common items found in various places throughout the user interface. It also discusses some of the more basic File, Settings, and Help menu items.

While editing data, you add a new item by using the Edit / New item menu item or clicking on the ⊞ button. To delete an item, use the Edit / Remove item menu item or click on the ⊠ button.

A number of these have a field called "Active". If the checkbox is checked, then this item will be used when syllabifying. If it is not checked, then it will not be used. This lets you experiment with various possibilities.

When the view contains a pane with rows and columns, you can click on a column header to sort by that column. You can even hold the Shift key down and click on another column header to get a secondary sort (or on a third column to get a tertiary sort). The default sort order is by Unicode code point. See section 2.5 for how to change the sort order. There are some tables which cannot be sorted because the user controls the order manually. In such cases (such as **CV Syllable Patterns** and **Sonority Hierarchy**), the sorting is disabled.

Asheninka will attempt to remember the location and size of the main window, table column widths, and most dialog boxes. So if you find that a dialog box is too small or too large, try resizing it (by dragging the edges of the window). Usually the next time you use that dialog box, it will be at the location and size you changed it to.

2.1 Creating a new project

Asheninka comes with a stock set of segments, etc. That is, when you create a new project, **Asheninka** will fill this new project with this stock set of segments, etc. You can then edit them, delete them, and add any needed items.

When you want to create a new project, use File / New Project menu item or click on the ▯ toolbar button. This brings up a standard "Save as…" dialog box so you can save this stock set of data as your new project. The expected file extension to use with all **Asheninka** files is ".ashedata" (for **Asheninka** data). Key the file name you want and press OK.

We suggest you create a directory called "My Asheninka" in your normal documents directory and put all your **Asheninka** projects in it. You may find it helpful to put each language project in its own directory.

2.2 Opening an existing project

When you already have an existing **Asheninka** project and you want to open it into **Asheninka**, use the File / Open Project... menu item or click on the ▣ toolbar button. This brings up a standard "File Open..." dialog box. Find the file you want and click on OK.

2.3 Saving a project

While **Asheninka** will automatically save your work about every 30 seconds, you can also use the File / Save Project menu item or click on the ▣ button in the toolbar.

2.4 Save a project as a new project

When you decide to save the current project with a different name, use the File / Save Project As... menu item. The resulting file will have all the data that the original project did except it will have a new name.

2.5 Setting writing system information for vernacular and analysis language display

Use the Settings menu item to set the writing system information for displaying the vernacular and/or the analysis languages. When you choose one of these, the dialog (which is patterned loosely after the writing system dialog in **FieldWorks Language Explorer**) has three tabs.

2.5.1 Font tab

The Font tab comes up by default. Click the "Choose font information" button to set the font family, font size, and font style.

Click the drop-down button on the color chooser to change the color from its default value of black.

2.5.2 General tab

There are thee items on this tab.

The language name value is for you to set, should you so wish. It will appear in the main window title if it is not blank.

The language code value is for you to set, should you so wish. It will appear in the main window title if the language name is not blank and this code is not blank.

The check box is for whether the language is written right-to-left or not. The default is not (i.e., left-to-right).[1]

---

[1]While we do offer this option, we expect that most right-to-left scripts do not normally use discretionary hyphens and/or are not necessarily orthographically designed to allow showing syllable breaks.

2.5.3 Keyboard tab

The Keyboard tab has a drop-down chooser. Click on it to show which keyboards are currently enabled and choose the one that is appropriate for the vernacular or analysis language you have chosen. **Asheninka** will automatically switch to this keyboard whenever a vernacular or analysis field is in focus.

Currently, this is fully available on Windows and mostly available on macOS.[2]

2.5.4 Sorting tab

The sorting tab has a drop-down chooser box that contains the following options:

1. Custom ICU rules
2. Default order
3. Same as another language
4. Use LDML file

These are described in the next sections.

2.5.4.1 Custom ICU rules

When this option is chosen, you can key in any custom rules using the ICU rule format.[3] If you use **FieldWorks Language Explorer** and have set up ICU rules there, you can copy and paste them into here.

As you type your rules, **Asheninka** attempts to parse the rules and whenever it finds an error, it gives a message in a box. It shows the approximate location of the error via an exclamation mark before the item. Please note that this appears as you type so if you are in the process of keying a rule, ignore the message until you finish the rule. If there is still a message, try and fix the rule. Whenever an error is found, the "OK" button is disabled so you cannot accidentally save a set of ICU rules with errors.

2.5.4.2 Default order

When the "Default order" option is chosen, **Asheninka** will sort items by using Unicode code points. That is, the order is based on the code number assigned to each Unicode character. For languages that use accents or non-A-Z characters, this often is incorrect. To fix it, please use one of the other options.

---

[2]On macOS, **Keyman** keyboards are handled as a unit. That is, if you have two or more of them, you will need to select the Keyman option. **Asheninka** will use whichever is the currently selected Keyman keyboard. This is an issue with macOS.

[3]See https://unicode-org.github.io/icu/userguide/collation/ and https://software.sil.org/fieldworks/wp-content/uploads/sites/38/2016/10/ICU-and-writing-systems.pdf.

2.5.4.3 Same as another language

Use this option if you need a sort order that is the same as a standard language. **Asheninka** comes with a large list of such languages and lists them in the "Language to use" drop-down chooser. The languages are given by their standard "locale" abbreviation.[4]

2.5.4.4 Use LDML file

A given language's LDML file contains any ICU rules as well as much more information.[5] **Asheninka** currently only uses the ICU rules, if any. If you have used **FieldWorks Language Explorer** and have set up a sort order for the language you are using **Asheninka** for and you happen to know where the LDML file is located,[6] then you can use this option to navigate to that file and select it. **Asheninka** will then find any ICU rules in it and use them.

2.6 Setting hyphenation parameters

When one exports syllabified words in one of three possible formats (as explained in section 12.2), one can also control three variables for each of the three formats:

1. The discretionary hyphen character sequence used.
2. The number of printable characters from the beginning of the word after which hyphenation starts.
3. The number of printable characters from the end of the word where hyphenation stops.

The default values for these can be changed by using the Settings / Hyphenation Parameters menu item. This then shows the three export options. Choosing one brings up a dialog box showing the current hyphenation parameter settings for this export method. You can change the values and then click on the OK button.

2.7 User interface language

You can set the user interface language by using the Settings / Change the interface language menu item. This brings up a dialog box showing the current interface language in a drop down chooser. Click on the chooser's drop down button to see other interface language choices. The choices given use the name of the language in the current interface language followed by the name of the language in its own language. So if the current interface language is English, then it will show "Spanish (español)" as an option; if the current interface language is Spanish, it will show English as "inglés (English)". When you click on the "OK" button, the program will "flash" and show with the new user interface language.

---

[4]See https://icu4c-demos.unicode.org/icu-bin/locexp/locexp for a list.
[5]See http://www.unicode.org/reports/tr35 for details.
[6]In a typical Windows installation, the LDML files are located at C:\ProgramData\SIL\FieldWorks\Projects*your_project_name*\WritingSystemStore.

The current version has English, French,[7] and a rough, most likely often inaccurate version of Spanish. Any corrections to any of the languages are welcome.

2.8 Help menu items

Currently, there are five Help menu items:

1. User Documentation which shows this document.
2. Suggested Steps which has some suggestions on how to be effective when using **Asheninka**.
3. Introduction to syllabification which has some general discussion of syllabification issues.
4. Hammond 1997 which opens Hammond (1997) in your default PDF viewer. If you plan to use the Optimality Theory approach, we highly recommend you read at least sections 4-7 (on pages 5-18) before doing so.
5. About Asheninka which has some standard information about the current version of **Asheninka**.

2.9 Status bar

The bottom row of the window (known as the status bar) contains the current date (at the left). Usually it also contains some numbers at the right. The rightmost pair of numbers is the current index for that view, a slash, followed by the total number of items in that view.

For word views, there are three sets of numbers. From the left they are:

1. The number of predicted parses, a slash, the total number of words, and the percentage for that ratio.
2. The number of predicted parses which equal the correct parse, a slash, the total number of words, and the percentage for that ratio.
3. The current index, a slash, and the total number of words.

The first two potentially change whenever you parse all words (by using the Parser / Parse all Words menu item or clicking on the ⌷ tool bar button). You can use the first set to help you see potential progress. The second can help you compare how your implementation of a particular approach is doing compared to some other approach which has been used to set the correct parse information (see section 10).

---

[7]Many thanks to the folks at Mission Assist for what is included. More help is needed to complete the French interface.

3 Approaches to syllabification

Asheninka offers six different syllabification algorithms or approaches. These are covered in sections 4–9.

The various approaches or algorithms are explained and illustrated in the "Introduction to Syllabification" document. You can read this document by clicking here or by using the Help / Introduction to syllabification menu item.

4 CV pattern approach

In the CV pattern approach, you need to define the following items:

1. segments (section 4.1)
2. natural classes (section 4.2)
3. syllable patterns (section 4.3)
4. grapheme natural classes (section 4.4)
5. environments (section 4.5)

After that, you import a word list (see section 12) or enter a list of words by hand. Then you use the Parser / Parse all Words menu item or click on the ⚏ tool bar button. This will apply the algorithm of the CV pattern approach to all the words. You can then see the results in the **CV Words** view. See sections 4.6 and 4.8 for ideas on how to check the results, among other things.

The main "game" to play with the CV pattern approach in **Asheninka** is to adjust the segment inventory (including graphemes and their environments), natural classes, and syllable patterns so that one gets most, if not all, words to syllabify correctly.

In rough terms, it tries to find all possible sequences of segments in the word. If it can do that, it then seeks to find a sequence of natural classes that covers the entire sequence of segments. Finally, it tries to find a sequence of syllable patterns that covers the entire sequence of natural classes. See appendix A for details.

We now give more information on the various views available in the user interface for the CV pattern approach.

4.1 CV segment inventory

Make sure that what is in the **Segment Inventory** view covers all the segments you have in the orthography. **Asheninka** tries all graphemes which match at the current point it is looking at in the left-to-right sweep. It tries the longest match first, but also applies any environments associated with the grapheme.

In the "Segment" field, key a letter or letters that will help you remember this segment. For example, a low central unrounded vowel might be keyed as **a** while a voiceless alveopalatal affricate might be keyed as $\widehat{\text{tʃ}}$ or as **ch**.

The "Description" field is for your benefit. Key whatever helps you (and anyone looking at your data) know what the segment is. For example, you could key its phonetic or phonological description.

In the "Graphemes" field, key all the ways the particular segment appears in your orthography, including any upper case forms. Separate each one by a space. When you leave this field (e.g., by clicking in the "Description" field or by pressing the **Tab** key or using **Shift-Tab**), **Asheninka** automatically shows the graphemes in the table at the bottom. You can choose to disable one or more of the graphemes by clicking on the check box in the table at the beginning of the row for the grapheme. You can also select one or more environments where the given grapheme is valid by clicking on the chooser button at the end of the row (see section 4.5 and also section 4.4 for more on environments). The chooser button looks like ⬚ . Note that whenever a given grapheme can occur as more than one segment, then be sure to include the appropriate environments for the grapheme in every such segment.

To remove a grapheme from the table, merely remove it from the "Graphemes" field. Note that when you uncheck a grapheme check box at the beginning of the row, **Asheninka** will automatically remove that grapheme from the "Graphemes" field. When you check that check box, **Asheninka** will automatically add it back to the "Graphemes" field. Thus, you can uncheck the box to see what might happen if this grapheme is no longer present. This can be useful when testing environments to condition a grapheme.

It is possible to have the same grapheme occur for more than one segment. If you do this, you may well want to add environments for when each grapheme is valid for a given segment.

4.2 CV natural classes

Make sure that in the **CV Natural Classes** view, every segment is in a natural class. A given segment may be in more than one class.

The "Name" field is for you to define how this natural class will appear in syllable patterns. We suggest using something short and if there is more than one character in it, make the first one be capitalized and the rest be in lower case.[8] While it is possible to use the same name more than once, it will most likely turn out to be confusing. So we recommend you make the name be unique.

The "Description" field is for your use to document anything special about this natural class.

The "Segment or Natural Class" field has a chooser button on the far right which looks like ⬚ . Click on it to bring up a dialog box which lets you select segments and/or natural classes which belong to this natural class. Note that it is possible to insert one entire natural class within another natural class. For example, if you have a class of nasal consonants, you can include that class within another class that has all consonants.

---

[8]This is not required, it is just a suggestion.

4.3 CV syllable patterns

Use the **CV Syllable Patterns** view to create CV syllable patterns that cover the kinds of syllables you think the language has. Be sure to allow for vowel-initial syllables if there are vowel-initial syllables in the language.

The "Name" field is for you to define a name for this syllable pattern. We suggest using something short. The "Description" field is for your use to document anything special about this syllable pattern.

The "Natural Classes" field shows the sequence of any natural classes this pattern consists of. It has a chooser button on the far right which looks like ⊡ . Click on it to bring up a dialog box which lets you select natural classes which constitute this syllable pattern. This chooser consists of one or more drop-down boxes. Click on the drop-down arrow in the box. It will show you the list of possible natural classes to choose from. In addition, if this is the very first drop-down box or the last one currently being shown, it will also include "Word boundary". When you choose "Word boundary", the pattern shown above the drop-down box(es), will show "#" to indicate a word boundary.

Finally, the order in which the syllable patterns occur is important, as mentioned in section 4. Recall that with the CV pattern approach, as **Asheninka** parses a word, it tries to match the patterns in the order in which they appear. You control the order of the syllable patterns by clicking on a pattern in the middle pane and then using the up and down arrows to change its order. Note that the normal way of sorting by a column via clicking on a column header is disabled for this view.

4.4 Grapheme natural classes

When your orthography is such that you need to condition some graphemes by one or more environments and you want to capture a generalization about which graphemes can occur around that grapheme via a natural class, then create one or more grapheme natural classes. Note that these are not the same as CV natural classes. The grapheme natural classes are based on the set of graphemes defined in all the segments. The CV natural classes, on the other hand, are defined based on the segments in the segment inventory. The first can be used to determine if a given grapheme is valid for its segment in a given orthographic environment. The second is used to define a class for a segment during the parsing process of a word into syllables.

The "Name" field is for you to define how this grapheme natural class will appear in environments. We suggest using something short and if there is more than one character in it, make the first one be capitalized and the rest be in lower case. The name should be unique among all the grapheme natural classes. If two or more grapheme natural classes share the same name, then any environment which refers to such a grapheme natural class will always use the first one **Asheninka** happens to find.

The "Description" field is for your use to document anything special about this grapheme natural class.

The "Grapheme or natural class" field has a chooser button (⬜) at the right edge. Clicking on this brings up a chooser where you can select the graphemes and/or other grapheme natural classes that should belong to the grapheme natural class you are defining. Note that it is possible to insert one entire grapheme natural class within another grapheme natural class. For example, if you have a class of nasal consonants, you can include that class within another class that has all consonants.

4.5 Environments

When your orthography is such that you need to condition some graphemes by one or more environments, you add the environments here. Besides the usual Active check box, there are three fields.

The "Name" field is for you to give a short name for this environment.

The "Description" field is for your use to document anything special about this environment, including why it is needed.

The "Environment" field is where you key the environment. Any error message will appear below the Active check box as you type. (When you create a new environment, this field contains "/ _ " which causes an error message of A class or a grapheme is missing; detected at or about position 6. This is because the environment is not valid until it has at least one grapheme, grapheme natural class, or word boundary symbol.)

The content of the "Environment" field follows a special notation. This notation is one that is reminiscent of what is used in many generative-style rules. The basic rules of thumb are:

(1)   1. Begin each environment with the forward slash character /
2. The location of the grapheme itself is indicated by an underscore character _
3. Any letters or grapheme natural classes that must come before the grapheme are typed in before this underscore character. Type them in the order in which they must appear.
4. Any letters or grapheme natural classes that must come after the grapheme are typed in after this underscore character. Type them in the order in which they must appear.
5. Any letter is indicated by typing it the way it appears in the practical orthography.
6. Any grapheme natural classes are indicated by
   a. typing a left square bracket [,
   b. typing the name of the grapheme natural class,[9] and
   c. then typing a right square bracket]

---

[9] This is another reason why you should use unique names for grapheme natural classes. If you have two or more grapheme natural classes with the same name, it is not clear which one you mean. **Asheninka** will automatically select one, but it may not be the one you intended.

When you type a left square bracket, a drop-down box will appear which has the list of grapheme natural class names. You can click on this drop-down box and then click on the name you want to use.[10]

7. Optional letters or grapheme natural classes are indicated by
   a. typing an opening parenthesis (,
   b. typing the letter or grapheme natural class as above, and
   c. then typing a closing parenthesis)
   d. Note that one should not nest optional items. The **Asheninka** parser will not handle these properly. You must enter each optional item after the other.

Example (2) gives some sample environments along with what they mean.

(2)

Environment	Meaning
/ m _	after an m
/ [V] _	after a vowel (assuming there is a grapheme natural class of vowels called V)
/ # i _	after a word initial i
/ # [V] _	after a word initial vowel (assuming there is a grapheme natural class of vowels called V)
/ [V] y _	after a vowel (assuming there is a grapheme natural class of vowels called V) and a y
/ _ i	before an i
/ _ [C]	before a consonant (assuming there is a grapheme natural class of consonants called C)
/ _ y #	before a y which is word final
/ _ [C] #	before a word final consonant (assuming there is a grapheme natural class of consonants called C)
/ m _ w	between an m and a w
/ [C] _ [C]	between two consonants (assuming there is a grapheme natural class of consonants called C)
/ ai _	after an a and an i
/ _ ai	before an a and an i
/ _ (a)i	before an optional a and an i; that is, either before ai or before i
/ _ ([C]) #	before an optional word final consonant (assuming there is a grapheme natural class of consonants called C); that is, either before a word final consonant or word finally

A given grapheme may have more than one environment, in which case the various environments are logically ORed with each other. That is, if any one of

---

[10]You can also press the **Tab** key two times to get to the drop-down box. Then use the **down-arrow** key to move to the one you want and press the **Enter** key. Please note that currently if you click on the drop-down box with the mouse and then use the **down-arrow** key, it will choose the first item in the list. I have yet to figure out how to avoid this poor behavior.

the environments for the grapheme are found, then the grapheme is considered to be valid (as far as its environments are concerned). For example, if a given grapheme can appear either before a consonant or word finally, then you can list both an environment for "before a consonant" and one for "before a word boundary." Example (3) shows what this might look like, assuming that you have a grapheme natural class of consonants with an abbreviation of C.

(3)
```
 / _ [C]        / _ #
```

4.6 CV words

In the **CV Words** view, the middle pane of the display shows all the words with four other columns:

1. **Comment**
2. **Predicted Syllable Breaks**
3. **Correct Syllable Breaks**
4. **Parser Result**

When you click on a word in the middle pane, the same fields are shown on the right where you can edit them. If the parse of the given word is successful, then a tree diagram of the parse is also shown as **Tree Diagram**. The W constituent is the word and the σ constituent indicates a syllable. The constituents below the syllable are the natural classes used. Below these are the segment and then its grapheme.

Please note that until you either manually add words to the project or until you import a set of words, the list will be empty. See section 12.1 for how to import a set of words.

The **Predicted Syllable Breaks** column/field contains the result of the last time you ran the **Parse all Words** tool. When it is empty, it means either that the parser has never been invoked or that the parser failed to produce a result. If the parser failed to produce a result, the **Parser Result** column/field should have some kind of explanation for where the parser ran into a problem (this will be in red).

When the **Predicted Syllable Breaks** column/field contains a result, it shows how this word was syllabified the last time you ran the **Parse all Words** tool, given the state of the segments, natural classes, and syllable patterns when it was invoked. When there is a result in this column/field, then the **Parser Result** column/field should have "Success" (in green).

Remember that you can click on a column header to sort the contents of the column. If you click on the **Parser Result** header, then any error messages should show at the top. You might find this useful when figuring out how to set the set of segments, natural classes, and syllable patterns to get a desired result.

The **Correct Syllable Breaks** column/field contains what you have indicated to be the correct syllabification for this word. It may be filled in if you imported words from a **Paratext** hyphenatedWords.txt file and that file indicated that this

word's hyphenation had been approved. See section 13.3 for how to clear the set of **Correct Syllable Breaks**. Otherwise, this column/field is expected to be blank until you either manually enter the correct syllabification or run the **Convert Predicted to Correct Syllabifcation** tool by using Tools / Convert predicted to correct syllabification menu item or clicking on the ▦ tool bar button.

The **Comment** field is for your use as needed. One possible scenario is to indicate why a given word does not parse. Another, if you are using IPA, is to enter how the word is spelled in the practical orthography.

When you run the **Convert Predicted to Correct Syllabifcation** tool, it brings up a dialog box listing all the words which have a non-empty **Predicted Syllable Breaks** field. See section 10 for more on this.
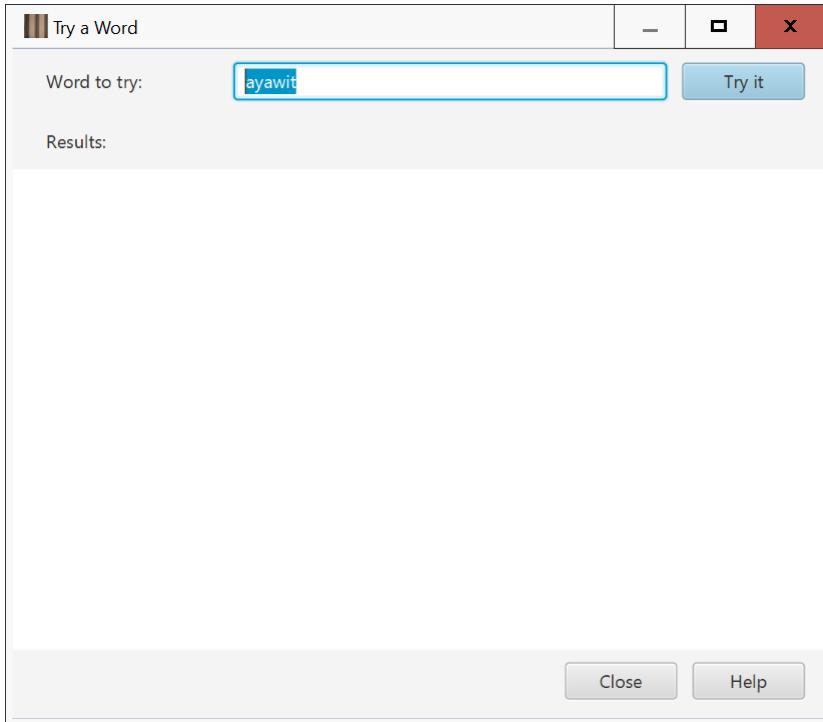
4.7 Predicted vs. correct CV words

The **Predicted vs. Correct CV Words** view shows any words which have both a predicted value and a correct value and, in addition, the two values differ. This is intended to give you a way to quickly see how the predictions of the current set of segments, natural classes, and syllable patterns differ from the expected results. In this view the predicted and correct words are aligned in pairs with the predicted syllabification immediately above the correct syllabification. This is an attempt to make it easier to see the differences between the two.

4.8 CV Try a Word

There are times when it may be difficult to know just why a given word is not syllabifying correctly. That's where the **Try a Word** dialog box can be useful. To see it, use the Parser / Try a Word menu item while in the CV pattern approach. This brings up a dialog box like the one shown in example (4). Unlike most dialog boxes, you can keep the **Try a Word** dialog box open while still editing other views in the main window. Unfortunately, the minimize button does not currently work so you cannot minimize the dialog, but you can drag it around and resize it. Further, the size and location of the **Try a Word** dialog box can be set independently for all six approaches; you can have several **Try a Word** dialog boxes open at the same time if you wish.

(4)

You can key a word to try in the text box. By default, **Asheninka** shows the current word selected in the **CV Words** view (if you are showing that view), the current word selected in the **Predicted vs. Correct CV Words** view (if you are showing that view) or the last word you used (or nothing if you've never used **Try a Word** before and are not showing the **CV Words** view).

When you either press the **Enter** key or click on the **Try it** button, **Asheninka** will try to parse the word in the text box and report the result in the Results portion of the dialog box.

For the CV pattern approach, there are at most three steps that are tried:

1. Parsing into segments.
2. Parsing segments into natural classes.
3. Parsing natural classes into syllables.

The results portion always shows the parsing into segments. If this succeeded, then it also shows the parsing of segments into natural classes. If this succeeded, it shows the result of parsing natural classes into syllables. Successful steps are shown in green while unsuccessful ones are shown in red. An example of a successful parse is shown in example (5).

(5)

Try a Word		_	▢	X

Word to try: | ayawit | | Try it

Results:

Syllabification of ayawit

Parsing into segments

Success!

Grapheme:	a	y	a	w	i	t
Segment:	a	y	a	w	i	t

Parsing segments into natural classes

Success!

V, C, V, C, {V,V+hi}, C

Close	Help

The parsing into segments part shows which grapheme was matched with what segment, Example (6) shows a case where the grapheme differs from the segment in shape.

(6)

Grapheme:	a	a	ng	u
Segment:	a	a	n	u

The parsing segments into natural classes part shows which natural classes were matched. Sometimes a given segment may be in more than one natural class. In that case, the possible natural classes are shown inside of curly braces as shown in example (7).

(7)   V, V, {C,N}, V

Scrolling the results window down shows the last part:

(8)

Try a Word   — ☐ ✕

Word to try: ayawit   Try it

Results:

Parsing natural classes into syllables

Success!

a.ya.wit

The details of the syllable patterns tried, in the order they were tried, is below. Click on the box with a plus sign in it to open that path. If a syllable pattern failed to match, it is in red. If a syllable pattern matched, but did not eventually lead to a successful parse, it is in blue. Finally, if a syllable pattern matched and led to a successful parse, it is in green. Note that successful parses are always the last item in the list because the algorithm quits after finding the first successful parse.

C, V

C, V, N

C, V, C

#, C, C, V

C, V, C, C, #

#, C, V, #

#, C, C, V+hi, C, #

V

Close   Help

When the parse succeeds, it shows the syllabification as well as a tree diagram indicating the structure of the word parsed into its syllables. The W constituent is the word and the σ constituent indicates a syllable. The constituents below the syllable are the natural classes used. Below these are the segment and then its grapheme.

In the last part, it shows which syllable patterns were tried in the order they were tried. Successful ones are shown in green (and begin with a small box with a plus sign in it). These will always be the last one shown because **Asheninka** stops looking whenever it finds a successful match. Failed ones are shown in red (and begin with a small box with a minus sign in it). Whenever a particular pattern matched the word at a particular point in the parsing process but it did not ultimately lead to a successful parse, it is shown in blue (and begins with a small box with a plus sign in it).

To examine a particular path, click on the small box with a plus sign in it. This will expand that path and show the next set of syllable patterns tried, in the order they were tried.
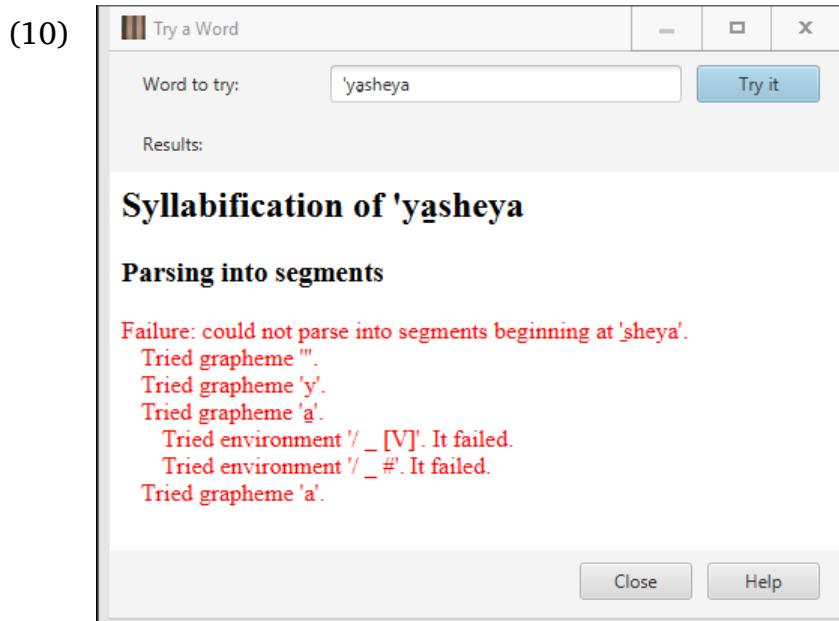
As mentioned above, whenever one of the three steps fails, there is an error message shown for that step (and no following steps will be tried).

For example, if a word contains a grapheme that is not defined in any segment, when trying to parse the word into segments, it will show a message such as what is in example (9).

(9)   Failure: could not parse into segments beginning at 'qap'.

In this case, the word was aanguqap but the q was not defined in any segment. Note that it shows where the problem was initially found.

It will also show the steps it took in trying to parse the word into segments, but failed as in example (10).

(10)

In this case, the grapheme a has two environments: either before a vowel or word finally. Neither succeeded.

As another example, if a word contains a segment that is not in any natural class, it will display a message such as what is shown in example (11).
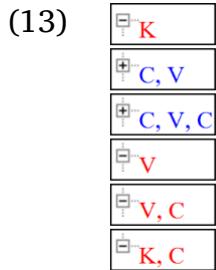
(11)   Failure: could not parse into natural classes; did find classes 'V, C' covering graphemes 'am'.

In the case of example (11), the word was ambu, but no natural class has the segment b in it. So when it tried to find a natural class for b, it failed. Notice that it does show what was found. (If the segment is the first item in the word, the found classes will be '' and the covered graphemes will be ''.)
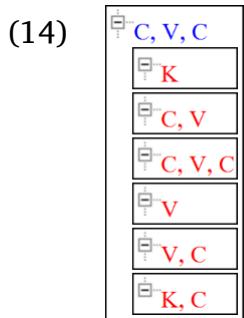
For the final step, if the sequence of natural classes could not be parsed into syllables per the ordered syllable patterns, then the following message is reported:

(12)   Failure: could not parse natural classes into syllables

What follows this is an explanatory paragraph as noted above for example (8) and then the list of syllable patterns tried. In one project, this looked liked what is in example (13) for the word yanjkuik.

(13)

| ⊟ K |
| ⊞ C, V |
| ⊞ C, V, C |
| ⊟ V |
| ⊟ V, C |
| ⊟ K, C |

Notice that none of the patterns are in green. Four are in red and two are in blue. Recall from above that this means that none of the red patterns matched the beginning of the sequence of natural classes (which was C, V, C, C, K, C). The two blue ones did match the beginning of the sequence, but did not ultimately lead to a successful parse. Example (14) shows what it looked like when opening the second blue pattern (C, V, C).

(14)

| ⊟ C, V, C |
| ⊟ K |
| ⊟ C, V |
| ⊟ C, V, C |
| ⊟ V |
| ⊟ V, C |
| ⊟ K, C |

Notice that all the embedded patterns are in red: none of them matched the beginning of the natural class sequence after removing the initial C, V, C part that had matched (i.e., with C, K, C).

5 Sonority hierarchy approach

In the Sonority hierarchy approach, you need to define the following items:

1. segments (section 5.1)
2. a sonority hierarchy via natural classes (section 5.2)
3. grapheme natural classes (section 5.5)
4. environments (section 5.6)

After that, you either use a existing word list, import a word list (see section 12) or enter a list of words by hand. Then you use the Parser / Parse all Words

menu item or click on the ⛫ tool bar button. This will apply the algorithm of the Sonority hierarchy approach to all the words. You can then see the results in the **Son. Hier. Words** view. See sections 5.3 and 5.7 for ideas on how to check the results, among other things.

The main "game" to play with the Sonority hierarchy approach in **Asheninka** is to adjust the segment inventory (including graphemes and their environments), and the sonority hierarchy (i.e., its ordered set of natural classes) so that one gets most, if not all, words to syllabify correctly. Depending on the needs of the language you are modeling, you may find that this algorithm is less than you need as noted in the "Introduction to Syllabification" document.

In rough terms, it tries to find all possible sequences of segments in the word. If it can do that, it then seeks to find a sequence of syllables based on the sonority hierarchy, where segments rise in sonority to the peak and then fall in sonority. See appendix B for details.

We now give more information on the various views available in the user interface for the Sonority hierarchy approach.

5.1 SH segment inventory

The **Segment Inventory** view works exactly like the **Segment Inventory** view for the CV pattern approach. See section 4.1 for details.

5.2 Sonority hierarchy

Use the **Sonority Hierarchy** view to create an ordered set of natural classes that constitute the sonority hierarchy you wish to use. More sonorous classes are above less sonorous classes.

The "Name" field is for you to define a name for this natural class. We suggest using something short. The "Description" field is for your use to document anything more specific about this class.

The "Segments" field shows the set of segments this class consists of. It has a chooser button on the far right which looks like ⬚ . Click on it to bring up a dialog box which lets you select the segments which constitute this class.

Asheninka checks to see two things:

1. If every segment appears in some class, and
2. if a given segment has already been used in some other class.

If either or both of these conditions exist, a box will appear showing which segments and or segment/class combinations are problematic. It is up to you to fix any problem. If a given segment is not in any class, then any word containing that segment will fail to syllabify. If a given segment is in more than one class, the syllabification algorithm will use the topmost class containing that segment.

Finally, the order in which the classes occur is important. The items are in order of sonority: the most sonorous is at top and the least sonorous is at the bottom. You control the order of the classes by clicking on a class in the middle pane and then using the up and down arrows to change its order. Note that the normal way of sorting by a column via clicking on a column header is disabled for this view.

5.3 Sonority hierarchy words

This works like the **CV Words** view for the CV pattern approach, except that the **Predicted Syllable Breaks** values, the **Parser Result** values, and the **Tree Diagram** are the result of the Sonority hierarchy approach algorithm. In the tree diagram, there is no intervening structure between the syllable and the segment. See section 4.6 for more on this view.

5.4 Predicted vs. correct Son. Hier. words

The **Predicted vs. Correct Son. Hier. Words** view shows any words which have both a predicted value and a correct value and, in addition, the two values differ. This is intended to give you a way to quickly see how the predictions of the current set of segments and sonority hierarchy differ from the expected results. In this view the predicted and correct words are aligned in pairs with the predicted syllabification immediately above the correct syllabification. This is an attempt to make it easier to see the differences between the two.

5.5 Grapheme natural classes

This works exactly like the **Grapheme Natural Classes** view for the CV pattern approach. See section 4.4 for details.
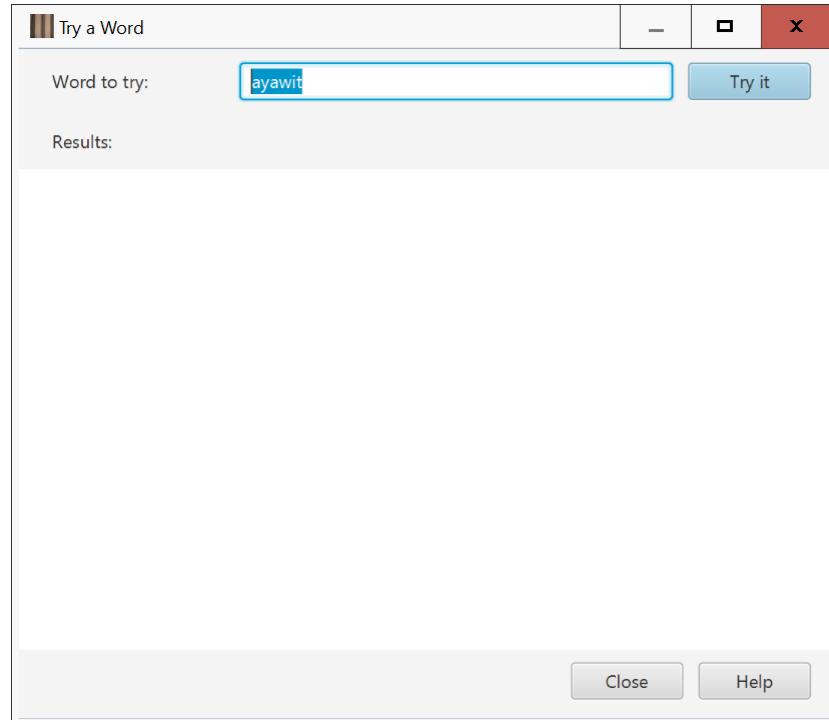
5.6 Environments

This works exactly like the **Environments** view for the CV pattern approach. See section 4.5 for details.

5.7 Sonority hierarchy Try a Word

There are times when it may be difficult to know just why a given word is not syllabifying correctly. That's where the **Try a Word** dialog box can be useful. To see it, use the Parser / Try a Word menu item. This brings up a dialog box like the one shown in example (15). Unlike most dialog boxes, you can keep the **Try a Word** dialog box open while still editing other views in the main window. Unfortunately, the minimize button does not currently work so you cannot minimize the dialog, but you can drag it around and resize it. Further, the size and location of the **Try a Word** dialog box can be set independently for all six approaches; you can have several **Try a Word** dialog boxes open at the same time if you wish.

(15)

▌▐ Try a Word	—	▢	✕

Word to try: [ayawit] [Try it]

Results:

[Close] [Help]

You can key a word to try in the text box. By default, **Asheninka** shows the current word selected in the **Son. Hier. Words** view (if you are showing that view), the current word selected in the **Predicted vs. Correct Son. Hier. Words** view (if you are showing that view) or the last word you used (or nothing if you've never used **Try a Word** before and are not showing the **Son. Hier. Words** view).

When you either press the **Enter** key or click on the **Try it** button, **Asheninka** will try to parse the word in the text box and report the result in the Results portion of the dialog box.

For the Sonority hierarchy approach, there are at most two steps that are tried:

1. Parsing into segments.
2. Parsing segments into syllables (via their natural classes).

The results portion always shows the parsing into segments. If this succeeded, then it also shows the parsing of segments into syllables. Successful steps are shown in green while unsuccessful ones are shown in red. An example of a successful parse is shown in example (16).

(16)

The parsing into segments part is exactly the same as it is for the CV pattern approach, although the tree diagram is simpler.

Scrolling the results window down shows the last part:

(17)

In the last part, it shows the sequence of segment pairs that were found along with their respective natural class and relationship. The **Relation** column uses mathematical symbols to indicate the relationship as given in example (18).

(18)

Relation	Meaning
<	Segment 1 is less sonorous than segment 2.
=	Segment 1 is equal in sonority to segment 2.
>	Segment 1 is more sonorous than segment 2.
!!!	The segment was not in any natural class.

The **Starts Syllable** column shows a syllable symbol (σ) when the algorithm began a new syllable.

As mentioned above, whenever one of the two steps fails, there is an error message shown for that step (and no following steps will be tried). See section 4.8 for examples where the segments could not be parsed.

If a word contains a segment that is not in any natural class, the sonority hierarchy results table will end with a row whose **Relation** column contains "!!!" in red and one of the offending segments will have "(No Natural Class)" after it as shown in example (19).

(19)

Segment 1	Relation	Segment 2	Starts Syllable
a (Vowels)	>	m (Nasals)	σ
m (Nasals)	!!!	b (No Natural Class)	σ

In the case of example (19), the word was ambu, but no natural class has the segment b in it. So when it tried to find a natural class for b, it failed

6 Onset-nucleus-coda approach

In the Onset-nucleus-coda approach, you need to define the following items:

1. segments (section 6.1)
2. a sonority hierarchy via natural classes (section 6.2)
3. syllabification parameters (section 6.3)
4. grapheme natural classes (section 6.9)
5. environments (section 6.10)

If needed to successfully parse your language, you may also need to define these items:

1. CV Natural Classes which are used in the next two (section section 6.4)
2. templates (section section 6.5)
3. filters (section section 6.6)

After that, you either use a existing word list, import a word list (see section 12) or enter a list of words by hand. Then you use the Parser / Parse all Words menu item or click on the ☰ tool bar button. This will apply the algorithm of the Onset-nucleus-coda approach to all the words. You can then see the results in the **ONC Words** view. See sections 6.7 and 6.11 for ideas on how to check the results, among other things.

The main "game" to play with the Onset-nucleus-coda approach in **Asheninka** is to adjust the segment inventory (including graphemes and their environments), the sonority hierarchy (i.e., its ordered set of natural classes) and the syllabification parameters so that one gets most, if not all, words to syllabify correctly. You may also need to write templates and/or filters to get some words to parse.

In rough terms, it tries to find all possible sequences of segments in the word. If it can do that, it then seeks to find sequences of onset, nucleus, and coda segments that can be placed in a syllable, modulo the parsing parameters. See appendix C for details.

We now give more information on the various views available in the user interface for the Onset-nucleus-coda approach.

6.1 ONC segment inventory

The **Segment Inventory** view is similar to the **Segment Inventory** view for the CV pattern approach. See section 4.1 for details. In addition to what the **Segment Inventory** view has, the **Segment Inventory** view also has three check boxes, one each for the following:

1. Can be in onset
2. Can be in nucleus
3. Can be in coda

Check the boxes for what positions the segment can be in.

6.2 Sonority hierarchy

The **Sonority Hierarchy** view works exactly like the **Sonority Hierarchy** view does for the Sonority hierarchy approach. See section 5.2 for details.

6.3 Syllabification parameters

The **Syllabification Parameters** view is where you set the parameters to control syllabification for your language. To read more about these parameters, you can click here or use the Help / Introduction to syllabification menu item.

The "Codas allowed" field is a check box. If codas are allowed, check the box. Otherwise, leave it unchecked.

The "Onset maximization" field is a check box. If onsets should be maximized, then check the box. Otherwise, leave it unchecked.

The "Onset principle" field has three radio buttons. One of them must be set. The three options are:

1. All but the first syllable must have an onset
2. Every syllable must have an onset
3. Onsets are not required

6.4 CV natural classes

The **CV Natural Classes** view is exactly the same as the **CV Natural Classes** view for the CV pattern approach. See section section 4.2 for details.

6.5 ONC templates

Templates can be used in two situations currently in **Asheninka** with the Onset-nucleus-coda approach:

1. For a nucleus, it requires the set of slots to be present.
2. For all others, it provides a way to override **Sonority Hierarchy** violations.

Note that syllable templates are not currently implemented.

When defining a template, you fill in the fields. The "Name" field is for you to remember this template.

Its "Description" field lets you define it more fully, perhaps including some example words as to why you added the template in the first place.

The "Type" field lets you select the constituent structure the template is for.

The "Slots" field is where you define the pattern for when the template will apply. You can use segment names or CV natural class names (see section 6.4) enclosed in square brackets (e.g., "[C]"). If the segment or the natural class can violate the **Sonority Hierarchy**, then key an asterisk (*) before the segment name or the opening square bracket. If the item is optional, enclose it within parentheses.

If you have a sequence of two or more items consisting of the same segment or class and at least one of the items is required while the rest are optional, then be sure to make the leftmost one(s) be required. (See the word final example in (20) below.)

Example (20) has some examples from an English project.

(20)	Type	Slots	Description
	Onset	*s [VoicelessNonCont] ([SonorantC])	Allow /s/ in an onset which violates the SSP
	Word final	*[Coronal] (*[Coronal]) (*[Coronal])	Appendix to allow for sɪksθs, etc.
	Word final	*[VoicelessNonCont] ([Coronal])	Word final appendix for /sp/ and /sk/

Finally, the order in which the templates occur is important. The items are in order of priority: the first to be tried (of a given type) is higher and the last to be tried (of that type) is lower. You control the order of the templates by clicking on a template in the middle pane and then using the up and down arrows to change its order. Note that the normal way of sorting by a column via clicking on a column header is disabled for this view.

6.6 ONC filters

Filters can be used in four situations currently in **Asheninka** with the Onset-nucleus-coda approach:

1. onset,
2. nucleus,
3. coda, and
4. rime.

Filters come in two varieties: fail and repair. When a filter is matched and is marked as fail, then the syllable parsing stops at that point and seeks other possibilities, if any. When the filter matches and is marked as repair, it will seek to fix up the syllable constituents around it to fix the situation. For example, for English, a word such as *Atlantic* (ætlæntɪk) will normally syllabify as æ.tlæn.tɪk but it needs to syllabify as æt.læn.tɪk. By adding an onset repair filter that matches the tl sequence, one can obtain the correct result.

When defining a filter, you fill in the fields. The "Name" field is for you to remember this filter.

Its "Description" field lets you define it more fully, perhaps including some example words as to why you added the filter in the first place.

The "Scope" field lets you select the constituent structure the filter has scope over.

The "Action" radio buttons let you specify whether the filter will fail when matched or attempt a repair.

The "Slots" field is where you define the pattern for when the filter will apply. You can use segment names or CV natural class names (see section section 6.4) enclosed in square brackets (e.g., "[C]"). If the segment or the natural class can violate the **Sonority Hierarchy**, then key an asterisk (*) before the segment name or the opening square bracket. If the item is optional, enclose it within parentheses. Just like with keying slots in templates, if you have a sequence of two or more items consisting of the same segment or class and at least one of the items is required while the rest are optional, then be sure to make the leftmost one(s) be required. (See the word final example in (20) above.)

If it is a repair filter, then key a vertical bar (|) at the point where the repair will occur.

Note that for onset-oriented repair filters, one can also key an underscore character (_) at the point before the first segment or class at the point where the onset

begins (i.e., just before the first segment or class in the onset). This enables you to give some context for the preceding syllable.

Example (21) has some examples from an English project. They are all repair filters. Example (22) has an example from Quiegolani Zapotec which makes use of the context in the preceding syllable ([Obs] is a class of obstruents).

(21)

Type	Slots	Description
Onset	[Non-stridentCoronal] | l	Disallow non-strident coronal before labial in an onset. Example: ætlæntɪk (*Atlantic*) should be æt.læn.tɪk, not æ.tlæn.tɪk.
Onset	[Stop] | [N]	Disallow stop/nasal in a onset. Example: æpniə (*apnea*) should be æp.ni.ə, not æ.pni.ə.
Onset	[Labial] | [Labial]	Disallow two labials in an onset. Example: ʃapwoɹn (*shopworn*) should be ʃap.woɹn, not ʃa.pwoɹn.

(22)

Type	Slots	Description
Onset	[V] [Sonorant] _ *[Obs] | ([Sonorant]) *[Obs]	Disallow an obstruent before another obstruent in an onset when the preceding syllable's rime contains a vowel and a sonorant. Example: baanske (*sadly*) should be baans.ke, not baan.ske.

Finally, the order in which the filters occur is important. The items are in order of priority: the first to be tried (of a given scope) is higher and the last to be tried (of that scope) is lower. You control the order of the filters by clicking on a filter in the middle pane and then using the up and down arrows to change its order. Note that the normal way of sorting by a column via clicking on a column header is disabled for this view.

6.7 ONC words

This works like the **CV Words** view for the CV pattern approach, except that the **Predicted Syllable Breaks** values, the **Parser Result** values, and the **Tree Diagram** are the result of the Onset-nucleus-coda approach algorithm. If the parse fails and some word structure was built, then this partial word structure is shown by **Tree Diagram**. In the tree diagram, a syllable may have an O (onset) and a required R (rime) constituent. A rime has an N (nucleus) and an optional C (coda) constituent. See section 4.6 for more on this view.

6.8 Predicted vs. correct ONC words

The **Predicted vs. Correct ONC Words** view shows any words which have both a predicted value and a correct value and, in addition, the two values differ. This is intended to give you a way to quickly see how the results of applying the Onset-nucleus-coda approach differ from the expected results. In this view the predicted and correct words are aligned in pairs with the predicted syllabification immediately above the correct syllabification. This is an attempt to make it easier to see the differences between the two.

6.9 Grapheme natural classes
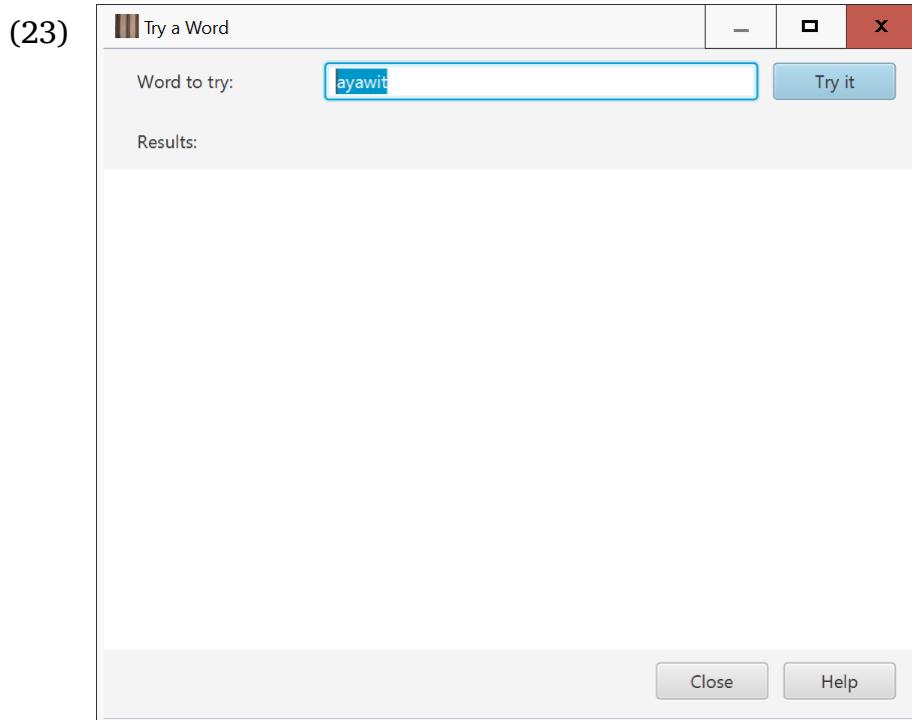
This works exactly like the **Grapheme Natural Classes** view for the CV pattern approach. See section 4.4 for details.

6.10 Environments

This works exactly like the **Environments** view for the CV pattern approach. See section 4.5 for details.

6.11 ONC Try a Word

There are times when it may be difficult to know just why a given word is not syllabifying correctly. That's where the **Try a Word** dialog box can be useful. To see it, use the Parser / Try a Word menu item. This brings up a dialog box like the one shown in example (23). Unlike most dialog boxes, you can keep the **Try a Word** dialog box open while still editing other views in the main window. Unfortunately, the minimize button does not currently work so you cannot minimize the dialog, but you can drag it around and resize it. Further, the size and location of the **Try a Word** dialog box can be set independently for all six approaches; you can have several **Try a Word** dialog boxes open at the same time if you wish.

(23)

You can key a word to try in the text box. By default, **Asheninka** shows the current word selected in the **ONC Words** view (if you are showing that view), the current word selected in the **Predicted vs. Correct ONC Words** view (if you are showing that view) or the last word you used (or nothing if you've never used **Try a Word** before and are not showing the **ONC Words** view).

When you either press the **Enter** key or click on the **Try it** button, **Asheninka** will try to parse the word in the text box and report the result in the Results portion of the dialog box.

For the Onset-nucleus-coda approach, there are at most two steps that are tried:

1. Parsing into segments.
2. Parsing segments into syllables (via the onset-nucleus-coda algorithm; see appendix C below).

The results portion always shows the parsing into segments. If this succeeded, then it also shows the parsing of segments into syllables. Successful steps are shown in green while unsuccessful ones are shown in red. An example of a successful parse is shown in example (24).

(24)
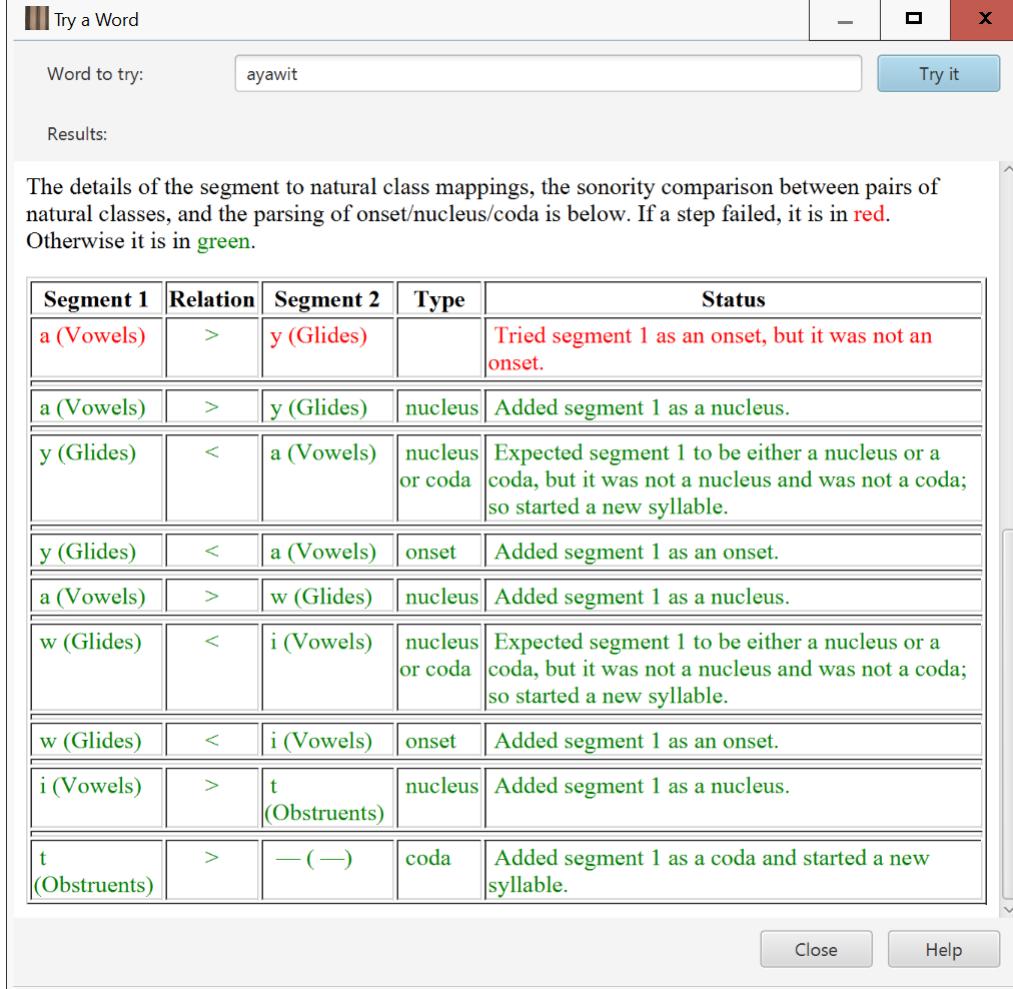
The parsing into segments part is exactly the same as it is for the CV pattern approach. If any constituent structure was built during the parse, the tree diagram is also shown. The `W` constituent is the word and the σ constituent indicates a syllable. A syllable may have an `O` (onset) and a required `R` (rime) constituent. A rime has an `N` (nucleus) and an optional `C` (coda) constituent. Below these are the segment and then its grapheme. If the parse succeeded, then the segments and their graphemes are shown in green. If the parse failed but some of the tree was built, then the segments and their graphemes are shown in red.

Scrolling the results window down shows the last part:

(25)

| Try a Word | | — □ X |

| Word to try: | ayawit | Try it |

Results:

The details of the segment to natural class mappings, the sonority comparison between pairs of natural classes, and the parsing of onset/nucleus/coda is below. If a step failed, it is in red. Otherwise it is in green.

Segment 1	Relation	Segment 2	Type	Status
a (Vowels)	>	y (Glides)		Tried segment 1 as an onset, but it was not an onset.
a (Vowels)	>	y (Glides)	nucleus	Added segment 1 as a nucleus.
y (Glides)	<	a (Vowels)	nucleus or coda	Expected segment 1 to be either a nucleus or a coda, but it was not a nucleus and was not a coda; so started a new syllable.
y (Glides)	<	a (Vowels)	onset	Added segment 1 as an onset.
a (Vowels)	>	w (Glides)	nucleus	Added segment 1 as a nucleus.
w (Glides)	<	i (Vowels)	nucleus or coda	Expected segment 1 to be either a nucleus or a coda, but it was not a nucleus and was not a coda; so started a new syllable.
w (Glides)	<	i (Vowels)	onset	Added segment 1 as an onset.
i (Vowels)	>	t (Obstruents)	nucleus	Added segment 1 as a nucleus.
t (Obstruents)	>	— (—)	coda	Added segment 1 as a coda and started a new syllable.

Close   Help

In the last part, it shows the sequence of segment pairs that were found along with their respective natural class and relationship. The **Relation** column uses mathematical symbols to indicate the relationship as given in example (26).

(26)

Relation	Meaning
<	Segment 1 is less sonorous than segment 2.
=	Segment 1 is equal in sonority to segment 2.
>	Segment 1 is more sonorous than segment 2.
!!!	The segment was not in any natural class.

The **Type** column shows the type (or state) the parser is using.
The **Status** column shows what the algorithm did at that part of the processing.

As mentioned above, whenever one of the two steps fails, there is an error message shown for that step. Depending on the error and the state of the parser at the time, more steps may or may not be tried. See section 4.8 for examples where the segments could not be parsed.

If a word contains a segment that is not in any natural class, the onset-nucleus-coda results table will end with a row whose **Relation** column contains "!!!" in red

and one of the offending segments will have "(No Natural Class)" after it as shown in example (27).

(27)

Segment 1	Relation	Segment 2	Type	Status
t (Obstruents)	!!!	u (No Natural Class)		Tried segment 1 as an onset, but the sonority hierarchy blocks it as an onset.
t (Obstruents)	!!!	u (No Natural Class)		Expected segment 1 to be a nucleus but it was not.

In the case of example (27), the word was tun, but no natural class has the segment u in it. So when it tried to find a natural class for u, it failed

7 Moraic approach

The Moraic approach, in many ways, is similar to the Onset-nucleus-coda approach. Like with the Onset-nucleus-coda approach, you need to define the following items:

1. segments (section 7.1)
2. a sonority hierarchy via natural classes (section 7.2)
3. syllabification parameters (section 7.3)
4. grapheme natural classes (section 7.9)
5. environments (section 7.10)

If needed to successfully parse your language, you may also need to define these items:

1. CV Natural Classes which are used in the next two (section 7.4)
2. templates (section 7.5)
3. filters section 7.6)

After that, you either use a existing word list, import a word list (see section 12) or enter a list of words by hand. Then you use the Parser / Parse all Words menu item or click on the ⚏ tool bar button. This will apply the algorithm of the Moraic approach to all the words. You can then see the results in the **Moraic Words** view. See sections 7.7 and 7.11 for ideas on how to check the results, among other things.

The main "game" to play with the Moraic approach in **Asheninka** is to adjust the segment inventory (including graphemes and their environments), the sonority hierarchy (i.e., its ordered set of natural classes) and the syllabification parameters so that one gets most, if not all, words to syllabify correctly. You may also need to write templates and/or filters to get some words to parse.

In rough terms, it tries to find all possible sequences of segments in the word. If it can do that, it then seeks to find sequences of segments that can be placed in a syllable with a mora-bearing segment at the "heart," modulo the parsing parameters. See appendix D for details.

We now give more information on the various views available in the user interface for the Moraic approach.

7.1 Moraic segment inventory

The **Segment Inventory** view is similar to the **Segment Inventory** view for the CV pattern approach. See section 4.1 for details. In addition to what the **Segment Inventory** view has, the **Segment Inventory** view also has a place to indicate how many moras the segment bears. Typically, consonants bear zero moras and vowels bear one or two.

7.2 Sonority hierarchy

The **Sonority Hierarchy** view works exactly like the **Sonority Hierarchy** view does for the Sonority hierarchy approach. See section 5.2 for details.

7.3 Syllabification parameters

The **Syllabification Parameters** view is where you set the parameters to control syllabification for your language. To read more about these parameters, you can click here or use the Help / Introduction to syllabification menu item. Like the Onset-nucleus-coda approach, the Moraic approach has the following three parameters:

The "Codas allowed" field is a check box. If codas are allowed, check the box. Otherwise, leave it unchecked.

The "Onset maximization" field is a check box. If onsets should be maximized, then check the box. Otherwise, leave it unchecked.

The "Onset principle" field has three radio buttons. One of them must be set. The three options are:

1. All but the first syllable must have an onset
2. Every syllable must have an onset
3. Onsets are not required

In addition, the Moraic approach also has two other parameters:

The "Maximum moras per syllable" field indicates the maximum number of moras a syllable in this language may have. Typically, this is one or two.

The "Use weight by position" field is a check box. If the language uses weight by position, check the box. Otherwise, leave it unchecked.

7.4 CV natural classes

The **CV Natural Classes** view is exactly the same as the **CV Natural Classes** view for the CV pattern approach. See section section 4.2 for details.

7.5 Moraic templates

Templates can be used in two situations currently in **Asheninka** for the Moraic approach:

1. For a syllable, it provides the pattern of optional and required slots that must be present for a well-formed syllable.
2. For onsets and word-initial and word-final, it provides a way to override **Sonority Hierarchy** violations.

Note that coda, nucleus and rime templates are not implemented for the Moraic approach because this approach does not model these constituents.

See section 6.5 for how to define a template.

7.6 Moraic filters

Filters can be used for onsets in **Asheninka** with the Moraic approach. While other filters are shown, only onset filters are used. See section 6.6 for more on using filters.

7.7 Moraic words

This works like the **CV Words** view for the CV pattern approach, except that the **Predicted Syllable Breaks** values, the **Parser Result** values, and the **Tree Diagram** are the result of the Moraic approach algorithm. If the parse fails and some word structure was built, then this partial word structure is shown by **Tree Diagram**. In the tree diagram, a syllable will have any onset consonants connected to an onset (O) constituent[11] and then to the syllable node. The syllable node will have any mora-bearing segments underneath a Greek mu (μ). See section 4.6 for more on this view.

7.8 Predicted vs. correct Moraic words

The **Predicted vs. Correct Moraic Words** view shows any words which have both a predicted value and a correct value and, in addition, the two values differ. This is intended to give you a way to quickly see how the results of applying the Moraic approach differ from the expected results. In this view the predicted and correct words are aligned in pairs with the predicted syllabification immediately above the correct syllabification. This is an attempt to make it easier to see the differences between the two.

7.9 Grapheme natural classes

This works exactly like the **Grapheme Natural Classes** view for the CV pattern approach. See section 4.4 for details.

---
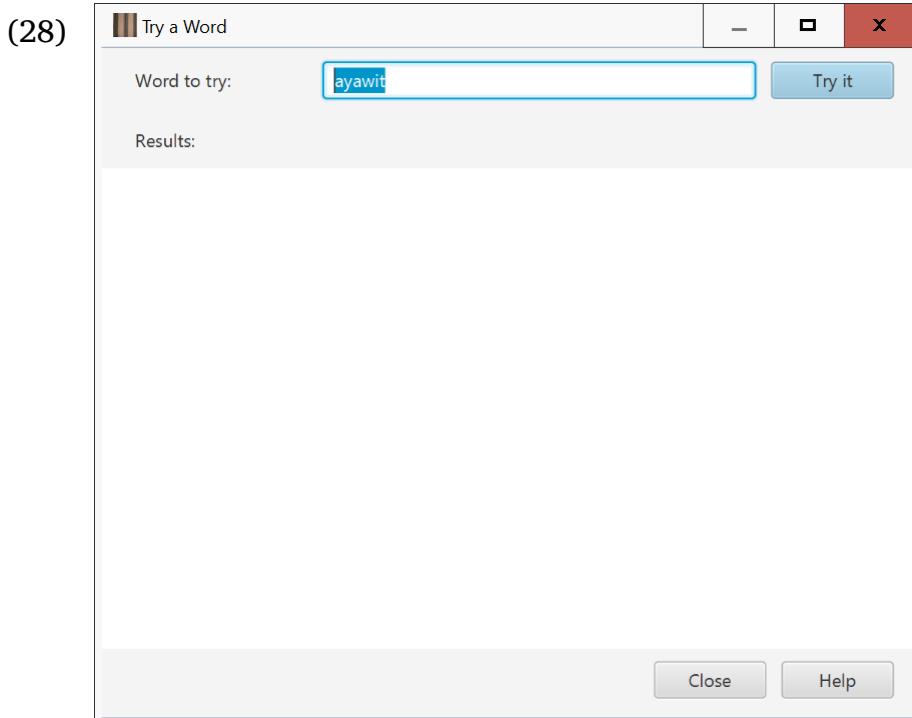
[11]The use of the O onset constituent is non-standard. We use it in light of how word games like pig-latin treat onsets as constituents.

7.10 Environments

This works exactly like the **Environments** view for the CV pattern approach. See section 4.5 for details.

7.11 Moraic Try a Word

There are times when it may be difficult to know just why a given word is not syllabifying correctly. That's where the **Try a Word** dialog box can be useful. To see it, use the Parser / Try a Word menu item. This brings up a dialog box like the one shown in example (28). Unlike most dialog boxes, you can keep the **Try a Word** dialog box open while still editing other views in the main window. Unfortunately, the minimize button does not currently work so you cannot minimize the dialog, but you can drag it around and resize it. Further, the size and location of the **Try a Word** dialog box can be set independently for all six approaches; you can have several **Try a Word** dialog boxes open at the same time if you wish.

(28)

You can key a word to try in the text box. By default, **Asheninka** shows the current word selected in the **Moraic Words** view (if you are showing that view), the current word selected in the **Predicted vs. Correct Moraic Words** view (if you are showing that view) or the last word you used (or nothing if you've never used **Try a Word** before and are not showing the **Moraic Words** view).

When you either press the **Enter** key or click on the **Try it** button, **Asheninka** will try to parse the word in the text box and report the result in the Results portion of the dialog box.

For the Moraic approach, there are at most two steps that are tried:

1. Parsing into segments.
2. Parsing segments into syllables (via the moraic algorithm; see appendix D below).

The results portion always shows the parsing into segments. If this succeeded, then it also shows the parsing of segments into syllables. Successful steps are shown in green while unsuccessful ones are shown in red. An example of a successful parse is shown in example (29).

(29)

The parsing into segments part is exactly the same as it is for the CV pattern approach. If any constituent structure was built during the parse, the tree diagram is also shown. The W constituent is the word and the σ constituent indicates a syllable. A syllable will have a μ constituent. Onset segments attach to an ○ category and other segments attach to a μ constituent. Below these are the segment and then its grapheme. If the parse succeeded, then the segments and their graphemes are shown in green. If the parse failed but some of the tree was built, then the segments and their graphemes are shown in red.

Scrolling the results window down shows the last part:

(30)

Try a Word

Word to try: ayawit [Try it]

Results:

The details of the segment to natural class mappings, the sonority comparison between pairs of natural classes, and the parsing of moras is below. If a step failed, it is in red. Otherwise it is in green.

Segment 1	Relation	Segment 2	Status
a (Vowels)	>	y (Glides)	Tried segment 1 as an onset, but it was not an onset.
a (Vowels)	>	y (Glides)	A syllable template (Syllable template "([C]) ([C]) [V] ([C]) ([C])") matched this: ay
a (Vowels)	>	y (Glides)	Added segment 1 with a mora
y (Glides)	<	a (Vowels)	Added the syllable to the word.
y (Glides)	<	a (Vowels)	A syllable template (Syllable template "([C]) ([C]) [V] ([C]) ([C])") matched this: yaw
y (Glides)	<	a (Vowels)	Added segment 1 as an onset directly to the syllable node
a (Vowels)	>	w (Glides)	Tried segment 1 as an onset, but it was not an onset.
a (Vowels)	>	w (Glides)	A syllable template (Syllable template "([C]) ([C]) [V] ([C]) ([C])") matched this: yaw
a (Vowels)	>	w (Glides)	Added segment 1 with a mora
w (Glides)	<	i (Vowels)	Added the syllable to the word.
w (Glides)	<	i (Vowels)	A syllable template (Syllable template "([C]) ([C]) [V] ([C]) ([C])") matched this: wit
w (Glides)	<	i (Vowels)	Added segment 1 as an onset directly to the syllable node
i (Vowels)	>	t (Obstruents)	Tried segment 1 as an onset, but it was not an onset.
i (Vowels)	>	t (Obstruents)	A syllable template (Syllable template "([C]) ([C]) [V] ([C]) ([C])") matched this: wit
i (Vowels)	>	t (Obstruents)	Added segment 1 with a mora
t (Obstruents)	>	— (—)	A syllable template (Syllable template "([C]) ([C]) [V] ([C]) ([C])") matched this: wit
t (Obstruents)	>	— (—)	Appended segment 1 to mora
t (—)	—	— (—)	Added the final syllable to the word.

[Close] [Help]

In the last part, it shows the sequence of segment pairs that were found along with their respective natural class and relationship. The **Relation** column uses mathematical symbols to indicate the relationship as given in example (31).

(31)

Relation	Meaning
<	Segment 1 is less sonorous than segment 2.
=	Segment 1 is equal in sonority to segment 2.
>	Segment 1 is more sonorous than segment 2.
!!!	The segment was not in any natural class.

The **Status** column shows what the algorithm did at that part of the processing. Note that if there is a syllable template and especially if that template includes one or more coda slots, then it is possible that the message will show more segments matching the syllable template than will actually end up being in the syllable.

   As mentioned above, whenever one of the two steps fails, there is an error message shown for that step. Depending on the error and the state of the parser at the time, more steps may or may not be tried. See section 4.8 for examples where the segments could not be parsed.

   If a word contains a segment that is not in any natural class, the moraic results table will end with a row whose **Relation** column contains "!!!" in red and one of the offending segments will have "(No Natural Class)" after it as shown in example (32).

(32)

Segment 1	Relation	Segment 2	Status
a (Vowels)	!!!	b (No Natural Class)	Tried segment 1 as an onset, but it was not an onset.

In the case of (32), the word was abay, but no natural class has the segment b in it. So when it tried to find a natural class for b, it failed

8 Nuclear projection approach

For the Nuclear projection approach, you need to define the following items:

1. segments (section 8.1)
2. a sonority hierarchy via natural classes (section 8.2)
3. syllabification parameters (section 8.3)
4. CV Natural Classes section 8.4)
5. NP rules (section 8.5)
6. grapheme natural classes (section 8.9)
7. environments (section 8.10)

If needed to successfully parse your language, you may also need to define NP filters (section 8.6).

   After that, you either use a existing word list, import a word list (see section 12) or enter a list of words by hand. Then you use the Parser / Parse all Words menu item or click on the ⧉ tool bar button. This will apply the algorithm of the Nuclear projection approach to all the words. You can then see the results in the

NP Words view. See sections 8.7 and 8.11 for ideas on how to check the results, among other things.

The main "game" to play with the Nuclear projection approach in **Asheninka** is to adjust the segment inventory (including graphemes and their environments), the sonority hierarchy (i.e., its ordered set of natural classes), the syllabification parameters, and the NP rules so that one gets most, if not all, words to syllabify correctly. You may also need to write NP filters to get some words to parse.

In rough terms, it tries to find all possible sequences of segments in the word. If it can do that, it then seeks to apply each rule in turn. If every segment in the word is in a syllable, then it considers the parse a success. See appendix E for details.

We now give more information on the various views available in the user interface for the Nuclear projection approach.

8.1 NP segment inventory

The **Segment Inventory** view works exactly like the **Segment Inventory** view for the CV pattern approach. See section 4.1 for details.

8.2 Sonority hierarchy

The **Sonority Hierarchy** view works exactly like the **Sonority Hierarchy** view does for the Sonority hierarchy approach. See section 5.2 for details.

8.3 Syllabification parameters

The **Syllabification Parameters** view is where you set the parameters to control syllabification for your language. To read more about these parameters, you can click here or use the Help / Introduction to syllabification menu item. Unlike some other approaches, the Nuclear projection approach has only one parameter.

The "Onset principle" field has three radio buttons. One of them must be set. The three options are:

1. All but the first syllable must have an onset
2. Every syllable must have an onset
3. Onsets are not required

The other two parameters are controlled by the NP rules:

• Whether codas are allowed or not is controlled by having coda-oriented rules or not having them.
• Onset maximization is controlled by ordering onset-oriented rules before any coda rules.

8.4 CV natural classes

The **CV Natural Classes** view is exactly the same as the **CV Natural Classes** view for the CV pattern approach. See section section 4.2 for details.

8.5 NP rules

NP rules drive the syllabification process in the Nuclear projection approach. Typically, at least two to five rules need to be defined. The first one must always be one that builds a nucleus and all the needed structure. It is usually built on vowels. The other rules add other segments to this.

When defining a NP rule, you fill in the fields. The "Name" field is for you to remember this rule.

Its "Description" field lets you define it more fully, perhaps including some example words as to why you added the rule if it is not one of the standard rules.

The "Affected" field lets you select the segment or natural class that the rule applies to.

The "Context" field lets you select the segment or natural class that must occur before or after the affected item.

The "Action" drop-down chooser lets you select the kind of action the rule has. There are five possible values as shown in example (33).

(33)

Action	Description
Attach	Attach the affected segment at the node level of the rule.
Augment	Augment the affected segment at the node level of the rule.
Build	Build the initial nodes. The affected segment will be at the N which will be in an N' node which will be in an N'' node which is the first item in a syllable.
Left adjoin	Adjoin the affected segment to the left of the context segment at the indicated node level.
Right adjoin	Adjoin the affected segment to the right of the context segment at the indicated node level.

The "Level" drop-down chooser lets you select the rule node level of the rule. There are four possible values as shown in example (34).

(34)

Level	Description
All	This indicates the rule will build all the levels.
N	This is the nucleus level.
N'	This is the rime level.
N''	This is the top level to which onsets are attached.

The "Obeys SSP" check box indicates whether or not the affected item and the context item must obey the Sonority Sequencing Principle.

Example (35) below lists the standard five rules. A dashed line indicates the item that being added.

(35)

Rule	Affected	Context	Action	Level	Diagram
Nucleus	[V]		Build	All	N" | N' | N | [V]
Onset	[C]	[V]	Attach	N''	N" ╱ ╲ ╱ N' ╱ | ╱ N ╱ | [C] [V]
Coda	[C]	[V]	Attach	N'	N' ╱ ╲ N ╲ | ╲ [V] [C]
Augmented onsets	[C]	[C]	Augment	N''	N" ╱ ╲ [C] [C]
Augmented codas	[C]	[C]	Augment	N'	N' ╱ ╲ [C] [C]

Notice that the level determines whether the affected segment is attached before (as an onset) or after (as a coda). That is, an N'' level means the affected item is attached before. An N' or N level means the affected item is attached after.

If a language does not have any codas, then you can either de-activate any coda-oriented rules or you can remove them.

If a language uses onset maximization, then order the rules so that any onset-oriented rules occur before any coda rules.

Asheninka only allows certain combinations of affected, context, action and level. Any other combinations will result in an error message stating what the allowed possibilities are. Some of these are summarized here:

- The first active rule must use a `Build` action.
- Only the `Build` action can use the `All` level.
- A context item is required for all but a `Build` action and a `Build` action must not have a context item.

Any adjunction rules will add the affected segment to the left (before) or to the right (after) depending on the action chosen.

Example (36) has a `Right adjoin` example from an English project. It is for the case of words like sɪksθ (*sixth*) and sɪksθs (*sixths*).

(36)

Rule	Affected	Context	Action	Level	Diagram
Word final adjoin	[Coronal]	[C]	Right adjoin	N''	N'' branching to N'' ([C]) and N'' (*[Coronal])

In this case, the SSP is not obeyed, which is indicated by the asterisk (*) before the affected segment. It uses the `N''` level because these segments are adjoined at the end of the word, like an appendix.

Finally, the order in which the rules occur is important. The items are in order of application: the first to be tried is higher and the last to be tried is lower. You control the order of the rules by clicking on a rule in the middle pane and then using the up and down arrows to change its order. Note that the normal way of sorting by a column via clicking on a column header is disabled for this view.

8.6 NP filters

The set of filters for the Nuclear projection approach is a completely different set of filters from those used by the Onset-nucleus-coda approach and the Moraic approach. NP filters can be used for onsets, codas, and rimes in **Asheninka** with the Nuclear projection approach. More precisely, NP filters are applied to rules whose Level is either `N'` or `N''`. Every NP filter is a fail filter. There are no repair filters in the Nuclear projection approach.

You define filters in a similar way to the filters for the Onset-nucleus-coda approach and Moraic approach. See section 6.6 for more on using filters.

One other difference is that for NP filters, you can use the underscore character (_) at the point before the first segment or class which is being attached or

augmented. For example, in an English IPA project, there is an NP filter which fails whenever an alveo-palatal is to be added before a sonorant. This is to handle cases like kæʃlɛs (*cashless*) which should be syllabified as kæʃ.lɛs, not as kæ.ʃlɛs and maɹʃlænd (*marshland*) which should be syllabified as maɹʃ.lænd, not maɹ.ʃlænd. As an initial attempt, we might write this filter as in example (37). This says that when we are about to add a segment that is a member of the Alveo-palatal class and is immediately followed by a segment which is a member of the Sonorant class, do not add it.

(37)   [Alveo-palatal] [Sonorant]

If we write this onset filter like this, we will discover that words such as ʃɹəb (*shrub*) will fail to syllabify (the filter blocks adding ʃ as an onset). To overcome this, we can write the filter as in example (38).

(38)   *[V] (*[Sonorant]) _ [Alveo-palatal] [Sonorant]

What comes before the underscore is the preceding context (a vowel with an optionally following sonorant). Since a word like ʃɹəb does not meet this context, the filter will not block the ʃ being added as an onset. Notice that for both kæʃlɛs and maɹʃlænd, the context is met and so the filter does block adding the ʃ as desired.

8.7 NP words

This works like the **CV Words** view for the CV pattern approach, except that the **Predicted Syllable Breaks** values, the **Parser Result** values, and the **Tree Diagram** are the result of the Nuclear projection approach algorithm. If the parse fails and some word structure was built, then this partial word structure is shown by **Tree Diagram**. In the tree diagram, a syllable always has onsets attached to N'', a nucleus attached to N and codas attached to N'. Any adjoined segments will show the adjoined level. See section 4.6 for more on this view.

8.8 Predicted vs. correct NP words

The **Predicted vs. Correct NP Words** view shows any words which have both a predicted value and a correct value and, in addition, the two values differ. This is intended to give you a way to quickly see how the results of applying the Nuclear projection approach differ from the expected results. In this view the predicted and correct words are aligned in pairs with the predicted syllabification immediately above the correct syllabification. This is an attempt to make it easier to see the differences between the two.

8.9 Grapheme natural classes
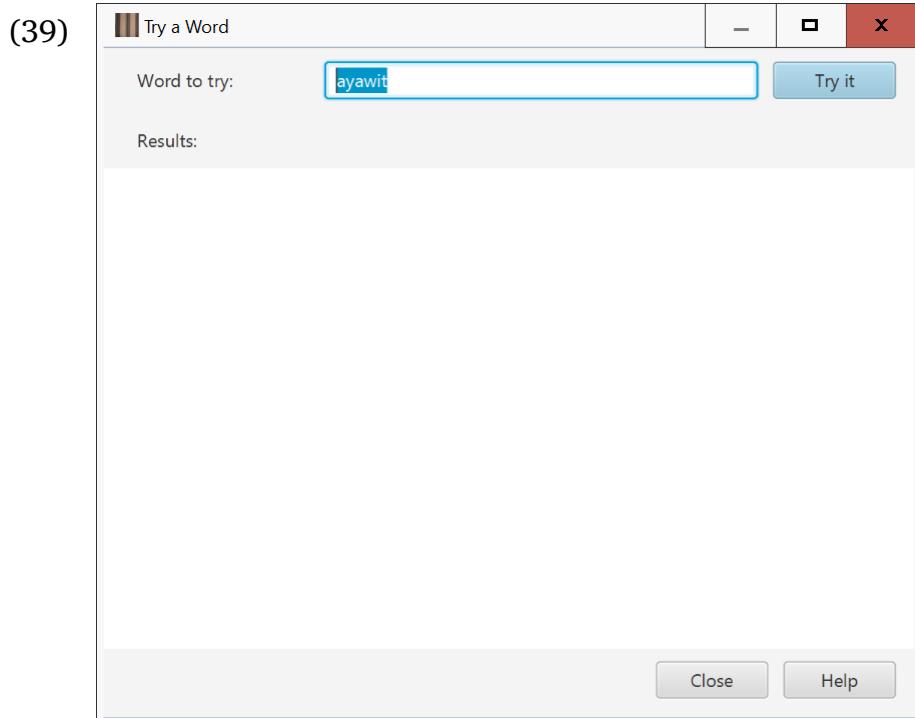
This works exactly like the **Grapheme Natural Classes** view for the CV pattern approach. See section 4.4 for details.

8.10 Environments

This works exactly like the **Environments** view for the CV pattern approach. See section 4.5 for details.

8.11 NP Try a Word

There are times when it may be difficult to know just why a given word is not syllabifying correctly. That's where the **Try a Word** dialog box can be useful. To see it, use the Parser / Try a Word menu item. This brings up a dialog box like the one shown in example (39). Unlike most dialog boxes, you can keep the **Try a Word** dialog box open while still editing other views in the main window. Unfortunately, the minimize button does not currently work so you cannot minimize the dialog, but you can drag it around and resize it. Further, the size and location of the **Try a Word** dialog box can be set independently for all six approaches; you can have several **Try a Word** dialog boxes open at the same time if you wish.

(39)

You can key a word to try in the text box. By default, **Asheninka** shows the current word selected in the **NP Words** view (if you are showing that view), the current word selected in the **Predicted vs. Correct NP Words** view (if you are showing that view) or the last word you used (or nothing if you've never used **Try a Word** before and are not showing the **NP Words** view).

When you either press the **Enter** key or click on the **Try it** button, **Asheninka** will try to parse the word in the text box and report the result in the Results portion of the dialog box.

For the Nuclear projection approach, there are at most two steps that are tried:

1. Parsing into segments.
2. Parsing segments into syllables (via the nuclear projection algorithm; see appendix E below).

The results portion always shows the parsing into segments. If this succeeded, then it also shows the parsing of segments into syllables. Successful steps are shown in green while unsuccessful ones are shown in red. An example of a successful parse is shown in example (40).

(40)

The parsing into segments part is exactly the same as it is for the CV pattern approach. If any constituent structure was built during the parse, the tree diagram is also shown. The `W` constituent is the word and the σ constituent indicates a syllable. A syllable will have `N''`, `N'`, and `N` constituents. Onset segments attach to an `N''` level, nucleus segments attach to an `N`, and coda segments attach to a `N'` level. Below these are the segment and then its grapheme. If the parse succeeded, then the segments and their graphemes are shown in green. If the parse failed but some of the tree was built, then the segments and their graphemes are shown in red.

Scrolling the results window down shows the next part:

(41)

| | Try a Word | — | ▢ | ✕ |

Word to try: ayawit   [Try it]

Results:

The details of the segment to natural class mappings and the application of rules is below. If a step failed, it is in red. Otherwise it is in green.

Status	Structure or SSP info
Applying rule: (Nucleus "Build nucleus")	
Built all nodes upon segment 'a'.	W σ N'' N' N a y a w i t a y a w i t
Built all nodes upon segment 'a'.	W σ σ N'' N'' N' N' N N a y a w i t a y a w i t
Built all nodes upon segment 'i'.	W σ σ σ N'' N'' N'' N' N' N' N N N a y a w i t a y a w i t

Applying rule: (Onset "Attach onset to nucleus")

[Close]   [Help]

Whenever a rule is applied, its name is shown in a row by itself. In (41), it is the first rule which builds a nucleus on each vowel. The resulting tree diagram is shown for each step. Note that in this case, the rule is applied as it matches an affected segment from left to right. `Augment` rules at the `N''` level and all `Left adjoin` rules are applied right to left. Scrolling down more, shows the next part:

(42)

This shows the onset rule being applied. The first row shows the checking of the SSP. It shows the sequence of segment pairs that were found along with their respective natural class and relationship. The **Relation** column uses mathematical symbols to indicate the relationship as given in example (43).

(43)

Relation	Meaning
<	Segment 1 is less sonorous than segment 2.
=	Segment 1 is equal in sonority to segment 2.
>	Segment 1 is more sonorous than segment 2.
!!!	The segment was not in any natural class.

Scrolling down further shows the next part:

(44)

This shows the result of the coda rule and that the other two rules did not apply to any segments.

As mentioned above, whenever one of the two steps fails, there is an error message shown for that step. Depending on the error and the state of the parser at the time, more steps may or may not be tried. See section 4.8 for examples where the segments could not be parsed.

If a word contains a segment that is not in any natural class, the results table will end with a row whose **Relation** column contains "!!!" in red and one of the offending segments will have "(No Natural Class)" after it as shown in example (45).

(45)

SSP failed.	Segment 1	Relation	Segment 2
	a (Vowels)	!!!	b (No Natural Class)

In this case, the word was abay, but no natural class has the segment b in it. So when it tried to find a natural class for b, it failed

9 Optimality Theory approach

For the Optimality Theory approach, you need to define the following items:

1. segments (section 9.1)
2. CV Natural Classes section 9.2)
3. OT Constraints (section 9.3)
4. OT Constraint Rankings (section 9.4)
5. grapheme natural classes (section 8.9)
6. environments (section 8.10)

After that, you either use a existing word list, import a word list (see section 12) or enter a list of words by hand. Then you use the Parser / Parse all Words menu item or click on the 🔣 tool bar button. This will apply the algorithm of the Optimality Theory approach to all the words. You can then see the results in the **OT Words** view. See sections 9.5 and 9.9 for ideas on how to check the results, among other things.

The main "game" to play with the Optimality Theory approach in **Asheninka** is to adjust the segment inventory (including graphemes and their environments), the OT Constraints, and the OT Constraint Rankings so that one gets most, if not all, words to syllabify correctly.

In rough terms, it tries to find all possible sequences of segments in the word. If it can do that, it then seeks to apply each constraint in turn, using a similar approach to that described in Hammond (1997). If every segment in the word is in a syllable, then it considers the parse a success. See appendix F for details. If you are familiar with Optimality Theory, please read at least sections 4-7 of Hammond (1997:5-18). He explains the need for implementing OT the way it is done here. For your convenience, we include this paper with **Asheninka** via the Help / Hammond (1997) menu item.

We now give more information on the various views available in the user interface for the Optimality Theory approach.

9.1 OT segment inventory

The **Segment Inventory** view works exactly like the **Segment Inventory** view for the CV pattern approach. See section 4.1 for details.

9.2 CV natural classes

The **CV Natural Classes** view is exactly the same as the **CV Natural Classes** view for the CV pattern approach. See section section 4.2 for details.

9.3 OT Constraints

OT Constraints are crucial for the syllabification process in the Optimality Theory approach.

When defining an OT Constraint, you fill in the fields. The "Name" field is for you to remember this constraint. (Normally OT Constraint names are shown using

small caps letters; unfortunately, we do not have a way to apply such a style auto-matically in the user interface.[12] We do use small caps in **OT Try a Word** because it is possible there.)

As described in Hammond (1997:15-18), each OT Constraint involves one or two sets of an affected element and structural options.

The "Affected element 1" field lets you select the first segment or natural class that the constraint applies to. You can also indicate if the element must occur word initially and/or word finally.

The "Structural Options 1" check boxes let you select the set of structural options the constraint will remove. See Hammond (1997:11ff) for more on the structural options; we use the same abbreviations here:

(46)

Structural Option	Meaning
o	Onset
n	Nucleus
c	Coda
u	Unsyllabified (or unparsed)

The "Affected element 2" field is optional. When it and the "Structural Options 2" fields are set, then both elements 1 and 2 must match before the constraint will apply.

The "Prune element 2" check box should be checked whenever the second set of structural options are to be removed instead of the first set.

Asheninka expects that the first affected element must be set and that at least one of the structural options for the first set must be checked. Similarly for the second set: if a second structural option is set, then the second affected element must be set, and vice versa. Whenever **Asheninka** detects something awry, it will show an error message in red, suggesting what is needed. When the settings are valid, a tree diagram of the constraint will appear.

9.4 OT Constraint Rankings

The set of OT Constraints need to be placed into a ranking. While you can have multiple rankings, only the one ordered first will be used when parsing. This way, you can experiment with different rankings to see how well the syllabification of that ranking works.

When defining a OT Constraint Ranking, you fill in the fields. The "Name" field is for you to remember this ranking.

Its "Description" field lets you define it more fully, perhaps including some ex-ample words as to why you added the ranking.

The "Constraint Ranking" field lets you select the order of the OT Constraints of this ranking. Clicking on the chooser button brings up a dialog showing the OT Constraints. The order in which the constraints occur is important. The items are in order of application: the first to be tried is higher and the last to be tried

---

[12]It is possible to use an overt small caps font to show these, but we have opted not to do that.

is lower. You control the order of the constraints by clicking on a constraint and then using the up and down arrows to change its order.

Similarly, the order in which the rankings occur is somewhat important. Only the first/topmost ranking will be used. You control the order of the rankings by clicking on a ranking in the middle pane and then using the up and down arrows to change its order. Note that the normal way of sorting by a column via clicking on a column header is disabled for this view.

When you create a new ranking, you will be shown a dialog box to select an existing ranking as a starting point for the new ranking. If you click on **Cancel**, the new ranking will be the same as the order of the list of constraints in the **OT Contraints** view.

9.5 OT words

This works like the **CV Words** view for the CV pattern approach, except that the **Predicted Syllable Breaks** values, the **Parser Result** values, and the **Tree Diagram** are the result of the Optimality Theory approach algorithm. If the parse fails and some word structure was built, then this partial word structure is shown by **Tree Diagram**. In the tree diagram, a syllable consists of the segments parsed into it. Above each segment (and its grapheme) is the structural option used. See section 4.6 for more on this view.

9.6 Predicted vs. correct OT words

The **Predicted vs. Correct OT Words** view shows any words which have both a predicted value and a correct value and, in addition, the two values differ. This is intended to give you a way to quickly see how the results of applying the Optimality Theory approach differ from the expected results. In this view the predicted and correct words are aligned in pairs with the predicted syllabification immediately above the correct syllabification. This is an attempt to make it easier to see the differences between the two.

9.7 Grapheme natural classes

This works exactly like the **Grapheme Natural Classes** view for the CV pattern approach. See section 4.4 for details.

9.8 Environments
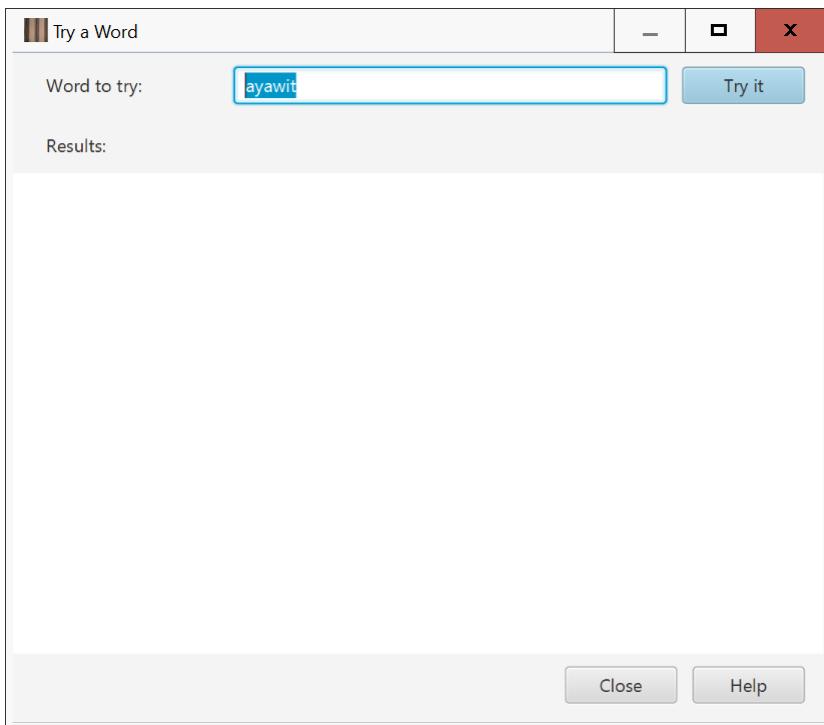
This works exactly like the **Environments** view for the CV pattern approach. See section 4.5 for details.

9.9 OT Try a Word

There are times when it may be difficult to know just why a given word is not syllabifying correctly. That's where the **Try a Word** dialog box can be useful. To see it, use the Parser / Try a Word menu item. This brings up a dialog box like

the one shown in example (47). Unlike most dialog boxes, you can keep the **Try a Word** dialog box open while still editing other views in the main window. Unfortunately, the minimize button does not currently work so you cannot minimize the dialog, but you can drag it around and resize it. Further, the size and location of the **Try a Word** dialog box can be set independently for all six approaches; you can have several **Try a Word** dialog boxes open at the same time if you wish.

(47)

Try a Word	—	◻	✕

Word to try: [ayawit] [Try it]

Results:

[Close] [Help]

You can key a word to try in the text box. By default, **Asheninka** shows the current word selected in the **OT Words** view (if you are showing that view), the current word selected in the **Predicted vs. Correct OT Words** view (if you are showing that view) or the last word you used (or nothing if you've never used **Try a Word** before and are not showing the **OT Words** view).

When you either press the **Enter** key or click on the **Try it** button, **Asheninka** will try to parse the word in the text box and report the result in the Results portion of the dialog box.

For the Optimality Theory approach, there are at most two steps that are tried:

1. Parsing into segments.
2. Parsing segments into syllables (via the optimality theory algorithm; see appendix F below).

The results portion always shows the parsing into segments. If this succeeded, then it also shows the parsing of segments into syllables. Successful steps are shown in green while unsuccessful ones are shown in red. An example of a successful parse is shown in example (48).

(48)

Try a Word

| Word to try: | ayawit | Try it |

Results:

Syllabification of **ayawit**

Parsing into segments

Success!

Grapheme:	a	y	a	w	i	t
Segment:	a	y	a	w	i	t

Success!

a.ya.wit

```
              W
      ┌───────┼───────────┐
      σ       σ           σ
      │      ╱╲          ╱╲
      n     o  n        o  n  c
      │     │  │        │  │  │
      a     y  a        w  i  t
      a     y  a        w  i  t
```

The parsing into segments part is exactly the same as it is for the CV pattern approach. If any constituent structure was built during the parse, the tree diagram is also shown. The W constituent is the word and the σ constituent indicates a syllable. A syllable will have one or more structural options. Below these are the segment and then its grapheme. If the parse succeeded, then the segments and their graphemes are shown in green. If the parse failed but some of the tree was built, then the segments and their graphemes are shown in red.

Scrolling the results window down shows the next part:

(49)

Each step in the parse consists of two candidate grids (see Hammond 1997:13), one on the left (the before candidate grid) and one on the right (the after candidate grid). Any structural options in the candidate grid on the left which are to be removed by the constraint being applied have a background color (green if the parse succeeded; red if it failed).

   Every parse begins with a candidate grid with all structural options available. The first "constraint" is the word initial/word final housekeeping one (see Hammond 1997:14) that ensures codas cannot occur word initially and onsets cannot occur word finally.

Whenever a constraint is applied, its name is shown above the candidate grid. Scrolling down more, shows the next part:

(50)
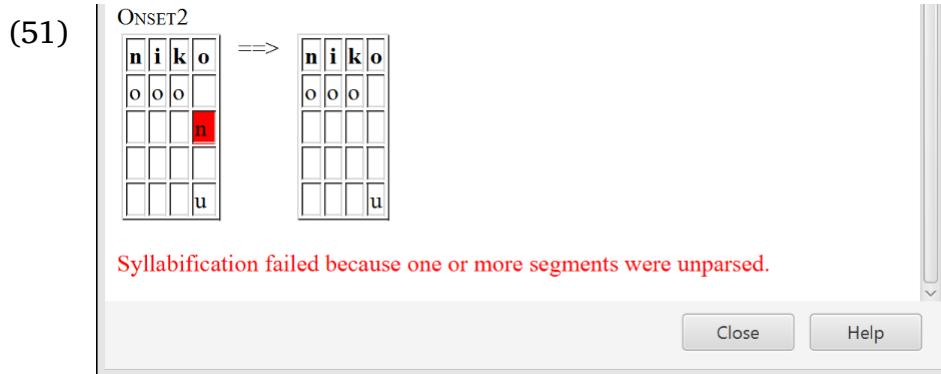
When only one structural option remains for each segment in the candidate grid, parsing stops. It then builds the syllables as it is able and shows each one in turn.

A parse fails whenever any of the following conditions occur:

- The only structural option remaining for some segment is "u" (i.e., it was unsyllabified or unparsed).
- All constraints were applied and some segment still has two or more structural options.

When a word fails to parse, there will be a message at the bottom to indicate why as shown in example (51).

(51)

Onset2

Syllabification failed because one or more segments were unparsed.

Close   Help

10 Converting predicted syllabification to correct syllabification
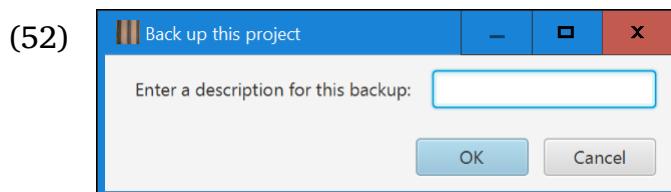
When you run the **Convert Predicted to Correct Syllabifcation** tool either by using the Tools / Convert predicted to correct syllabification menu item or by clicking on the ⛁ tool bar button, it brings up a dialog box listing all the words which have a non-empty **Predicted Syllable Breaks** field and that differ from any non-empty **Correct Syllable Breaks**. You can select which words should be converted by manually checking the box before them[13] or by clicking on the checkbox in the header row and choosing to "Select All", "Clear All", or "Toggle":

1. "Select All" checks every word.
2. "Clear All" unchecks every word.
3. "Toggle" makes every word that is checked be unchecked and every word that is unchecked be checked.

When you click on the OK button, **Asheninka** will copy the value in the **Predicted Syllable Breaks** field of every word that is checked to the corresponding **Correct Syllable Breaks** field.
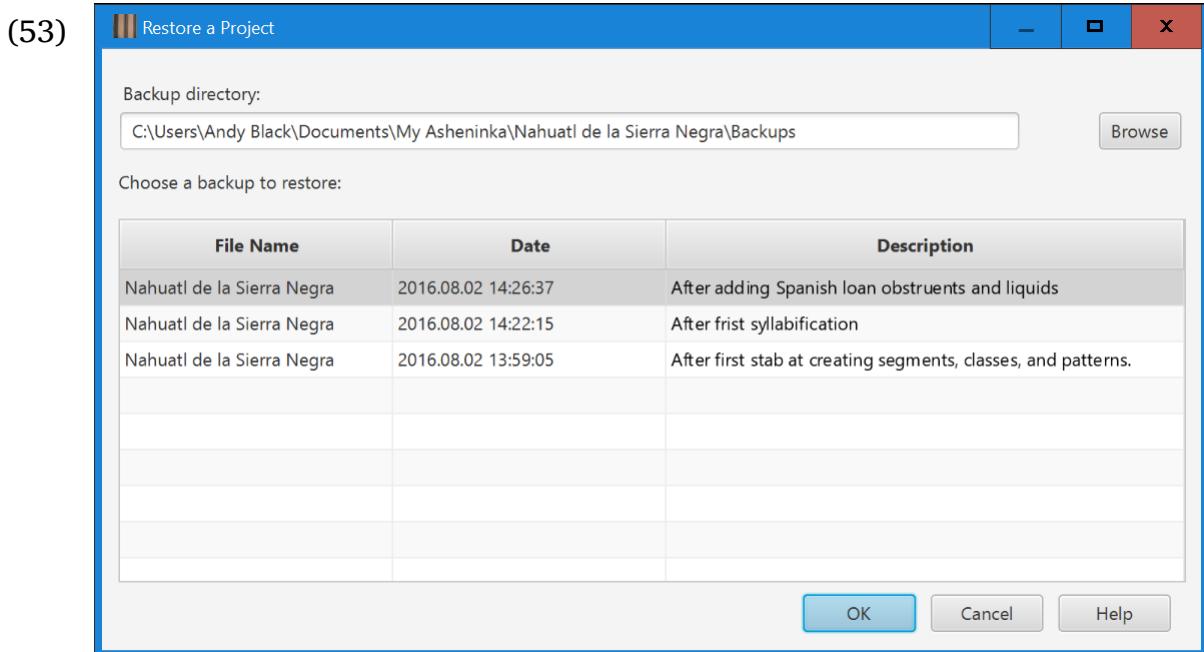
11 Project management

Feel free to make frequent (labeled) backups via the File / Project Management / Back up this project menu item. It will bring up a dialog box like the one shown in example (52). Please give it a descriptive label so you can later identify the state of your project when you made the backup. (You can key language data if you wish.) You won't see the label on the file name, though. Rather, you'll see it when you go to the File / Project Management / Restore a project menu item.

(52)

Back up this project

Enter a description for this backup:

OK   Cancel

---

[13]To check the box, either click in it or click in the row and press the space bar.

When you restore a project, you will see a dialog box something like the one shown in example (53).

(53)

The backups you have made for this project will be shown in the list. The content in the Description column is the description you keyed while making the backup.

The idea here is that you can make a backup, try something else (e.g., add a segment or class; or re-order the syllable patterns or add a new one, etc.), and see how it goes. If it's worse, just restore to the previous state. If it's better, make a backup of that and go on.

12 Importing and exporting words

When using **Asheninka**, of course, you need words to parse. Once they are parsed and you have set the correct syllabifications for them (see section 10), you may want to export the set of words to be used by a typesetting tool. Such a tool can use the syllabified words as a list of words with discretionary hyphens so that it can know when to hyphenate long words at ends of lines.

12.1 Import words

Asheninka allows you to import a list of words in the following forms:

1. A text file with one word per line
2. A word list exported from **Paratext**
3. The hyphenatedWords.txt file from **Paratext**
4. A word list exported from **FieldWorks Language ExplorerFieldWorks Language Explorer** (aka FLEx) (in tab-delimited form)

You can use any or all of these to add words to **Asheninka**'s list of words. A given word form will only be inserted once. By "word form" we mean that word pairs such as achto and Achto will be considered different since their capitalization is distinct.

To import a word list, use File / Import Words menu item and then choose the type of import to use. In each case, a standard file open dialog will appear. Find the file that has the format you need, choose it, and click on "Open". Depending on the number of words in the imported file, this may take a while to complete.

If you are working on a word list found in **Paratext**, we recommend using the hyphenatedWords.txt file method. This file includes both upper and lower case forms of words and also will include any hyphenation the **Paratext** user has manually approved. At least with some versions of **Paratext**, this file can be found in the "My Paratext Projects" folder and then in the folder of your project name.

Newly added words will appear in the words views with a **Parser Result** showing "Untested". This means that the parser has not yet been run on the word.

12.2 Export words

When you want to export the word list in **Asheninka** to some kind of typesetting tool, use the File / Export Hyphenated Words menu item. The list of hyphenated words can be exported in three formats:

1. Export for **InDesign** (simple list)
2. Export for **Paratext** (hyphenatedWords.txt)
3. Export for **XLingPaper** (hyphenations exception file)

Asheninka exports hyphenated words in the following way:

1. When the word has a value in **Correct Syllable Breaks**, that value is used.
2. If **Correct Syllable Breaks** is empty, then if there is a value in **Predicted Syllable Breaks** for the currently selected approach, it uses that value.
3. Otherwise, it does not export the word.

It also will replace the periods used to demarcate syllables with whatever you have set as the discretionary hyphen character(s) in the hyphenation parameters (see section 2.6). By default, **Asheninka** uses an equals sign for both **Paratext** and **InDesign** and a hyphen for the **XLingPaper** output.

Further, **Asheninka** will not insert a discretionary hyphen if it appears too close to the front or too close to the end of a word, depending on the hyphenation parameters set. By default, the **InDesign** export uses zero characters from the front and zero characters from the end (i.e., every potential discretionary hyphen position is used).[14] By default, both the **Paratext** and **XLingPaper** export methods limit discretionary hyphens from two characters from the front and two characters from the back.

---
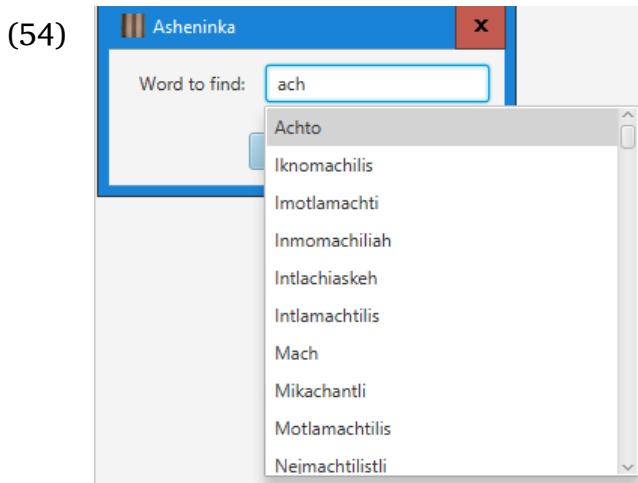
[14]This is because the **InDesign** program has its own way of letting the typesetter control where discretionary hyphens will be used.

13 Other tools

Asheninka also offers five other tools not previously covered.

13.1 Find a word

Use the Tools / Find Word menu item or key **Ctrl F** or click on the ▨ button on the toolbar to find a particular word. This brings up a dialog box which allows you to start typing the word you are looking for. As you type, it has a drop down area showing all words which contain the sequence of characters you have typed. If you are currently showing one of the Predicted vs. correct words views, then it only uses those words in this view. If you are showing any other view, it shows all words which match what you type. Note that it shows all words which have this sequence anywhere in them, not just at the beginning. For example, with one project when I typed "**ach**", what is in example (54) showed.

(54)

This can be very useful not only for finding a particular word, but also for looking for particular sequences of characters. A potentially better option for the latter, however, is described in section 14.
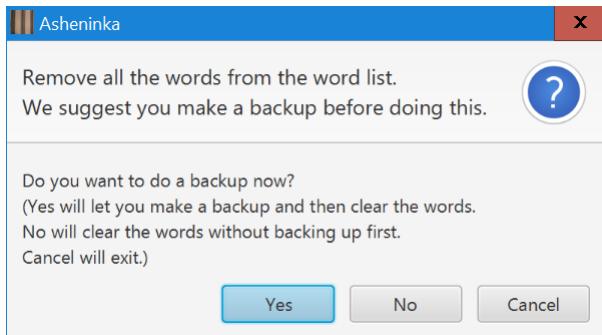
When your press the **Enter** key or click on the **OK** button, **Asheninka** will show the selected word in the approach you last used. The exception is if you currently selected any of the predicted vs. correct views, it will select the chosen word in that view.

If you happen to have any filters set on any word columns (see section 14) and the word you chose is not currently in the list of filtered words, then **Asheninka** will show a dialog box indicating that the word is hidden. You can remove all filters to show that word or you can cancel the process of finding that word.

13.2 Remove all words

Use the Tools / Remove all words menu item to completely clear the list of words in all of the words views. When you invoke this menu item, it will show the dialog box shown in example (55).

(55)

When you click on the **Yes** button, **Asheninka** will then show you the backup dialog box (see section 11). When you have done the backup, **Asheninka** will immediately remove all the words from the list of words.

If, instead, you click on the **No** button, **Asheninka** will immediately remove all the words from the list of words.
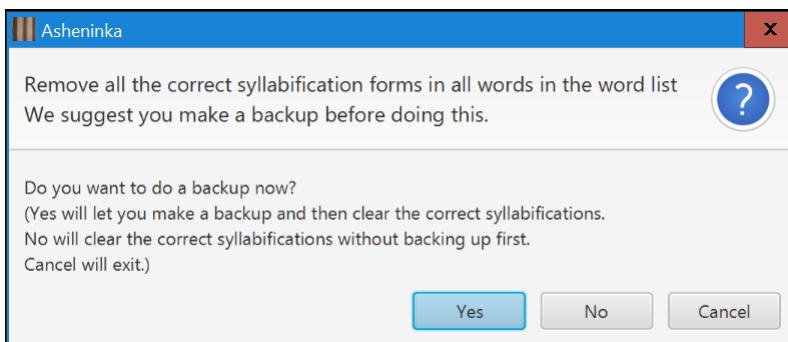
Note that when the words are removed, it will happen whether or not any of the words views are currently being shown. It usually happens very quickly.

If you click on the **Cancel** button (or close the dialog box using the X in the red area), the dialog box will exit and nothing will change.

13.3 Remove correct syllable breaks in all words

Use the Tools / Remove correct syllable breaks in all words menu item to completely clear the **Correct Syllable Breaks** column in all the words views. When you invoke this menu item, it will show the dialog box shown in example (56).

(56)

Like the dialog box for removing all words in section 13.2, you are asked whether or not to make a backup. When you click on the **Yes** button, **Asheninka** will then show you the backup dialog box (see section 11). When you have done the backup, **Asheninka** will immediately remove all the **Correct Syllable Breaks** forms from the list of words.
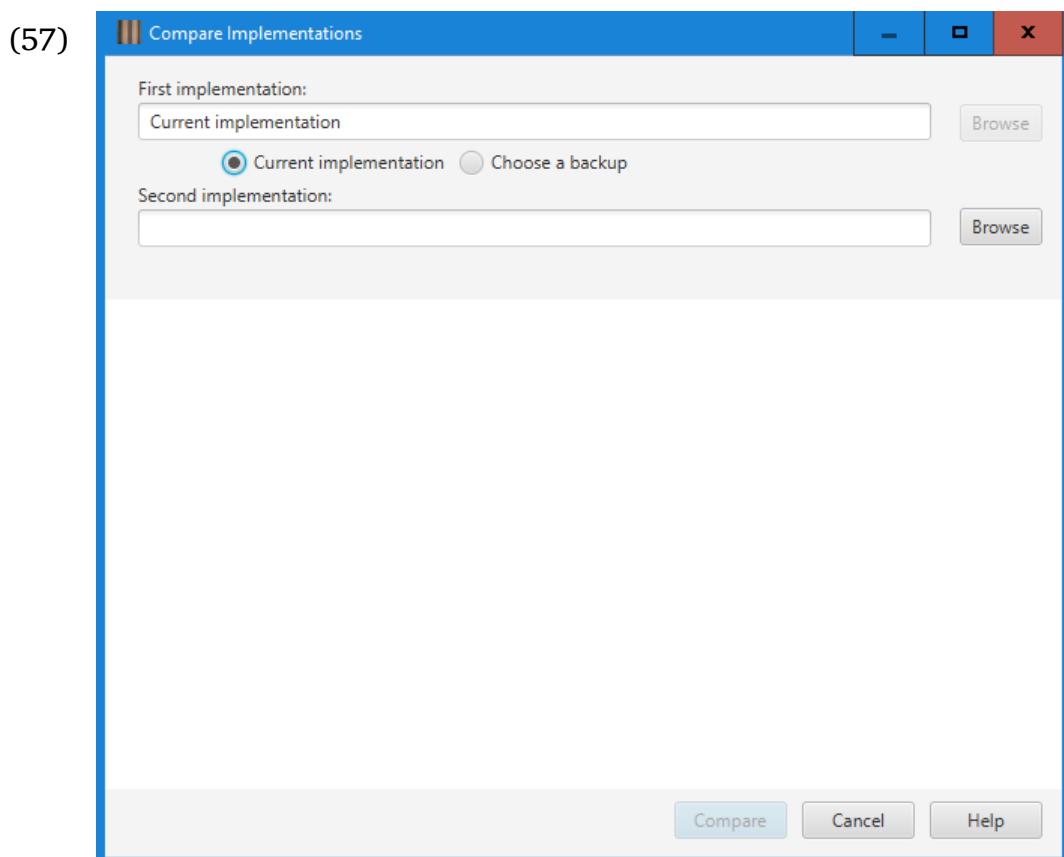
If, instead, you click on the **No** button, **Asheninka** will immediately remove all the **Correct Syllable Breaks** forms from the list of words.

Note that when the **Correct Syllable Breaks** forms are removed, it will happen whether or not any of the words views are currently being shown. It usually happens very quickly.

If you click on the **Cancel** button (or close the dialog box using the X in the red area), the dialog box will exit and nothing will change.

13.4 Compare two implementations

Asheninka offers a way to more easily see what some changes to an approach have on how the words are syllabified. Use the Tools / Compare Implementations menu item to do this. It brings up a dialog box which looks like what is in example (57).

(57)

You can compare the current implementation (i.e., what is showing in the user interface right now) with what is in a backup file (see section 11) or you can compare two different backups. As you can see in example (57), the current implementation is set by default. To compare two backups, click on the "Choose a backup" radio button.
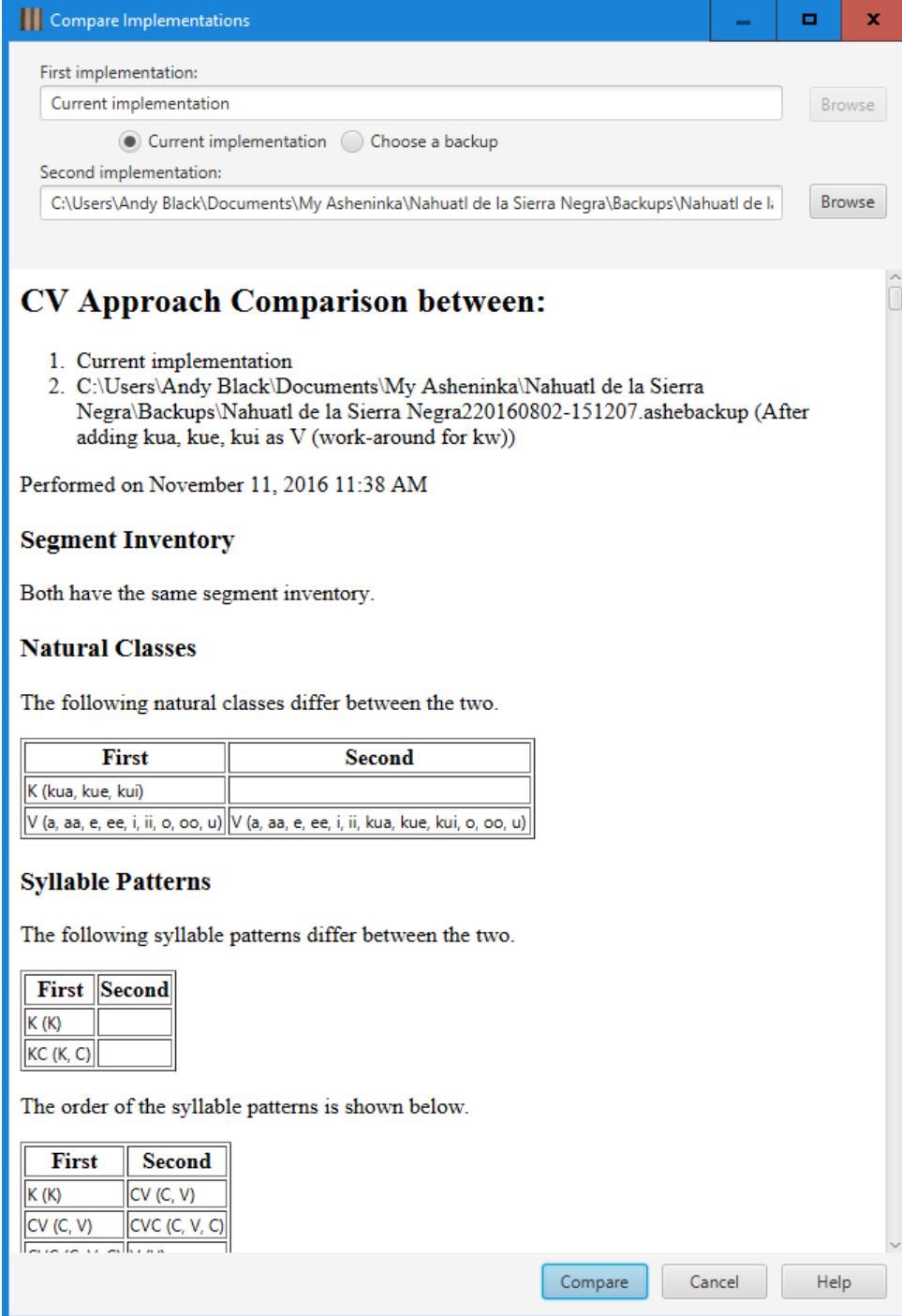
Whenever you need to select a backup file, click on the "Browse" button. This will bring up the "Restore a Project" dialog. Choose the backup you want and click OK.

When you have chosen the two implementations to compare, click on the "Compare" button. Depending on how many differences there are, the comparison process can take quite a long time so please be patient. When it is done, a report is shown in the bottom part of the dialog. The dialog box will stay visible until you close it (e.g., by clicking on the "Cancel" button). This lets you examine the report while still being able to make changes to the data in the user interface.

Which approach is used in the comparison is the current one being shown.

13.4.1 CV pattern approach comparison

When I did this for one project using the CV pattern approach, I got what is shown in example (58).

(58)

13.4.2 Sonority hierarchy approach comparison

When I did this for one project while showing the Sonority hierarchy approach, I got what is shown in example (59).

(59)

Compare Implementations — □ X

First implementation:

Current implementation   Browse

◉ Current implementation   ◯ Choose a backup

Second implementation:

C:\Users\Andy Black\Documents\My Asheninka\SonorityHierarchy\Backups\SHTsting20190129-132625   Browse

Sonority Hierarchy Approach Comparison between:

1. Current implementation
2. C:/Users/Andy Black/Documents/My Asheninka/SonorityHierarchy/Backups/SHTsting20190129-132625.ashebackup (Standard)

Performed on January 29, 2019 2:13 PM

Segment Inventory

Both have the same segment inventory.

Grapheme Natural Classes

Both have the same set of grapheme natural classes.

Environments

Both have the same set of environments.

Sonority Hierarchy

Both have the same set of natural classes in the sonority hierarchy.

The order of the sonority hierarchy is shown below.

First	Second
Vowels (a, e, i, o, u)	Vowels (a, e, i, o, u)
Glides (w, y)	Glides (w, y)
Nasals (m, n, ɲ)	Liquids (l, r)
Liquids (l, r)	Nasals (m, n, ɲ)
Obstruents (ch, d, f, g, h, k, p, s, sh, t, v, x, z)	Obstruents (ch, d, f, g, h, k, p, s, sh, t, v, x, z)
Voiced Obstruents (b, d, g, z)	Voiced Obstruents (b, d, g, z)

Words

The following words differ between the two.

First	Second	Syllabification
ahwiyalme		a.hwi.ya.lme
	ahwiyalme	a.hwi.yal.me

Compare   Cancel   Help

13.4.3 Onset-nucleus-coda approach comparison

When I did this for one project while showing the Onset-nucleus-coda approach, I got what is shown in example (60).

(60)

Compare Implementations ▯ ▢ ✕

First implementation:

Current implementation | Browse

◉ Current implementation ◯ Choose a backup

Second implementation:

C:\Users\Andy Black\Documents\My Asheninka\ONC\Backups\ONCTesting20190823-103204.a | Browse

ONC Approach Comparison between:

1. Current implementation
2. C:/Users/Andy Black/Documents/My Asheninka/ONC/Backups/ONCTesting20190823-103204.ashebackup (All but first has onset)

Performed on August 23, 2019 10:32 AM

Segment Inventory

Both have the same segment inventory.

Grapheme Natural Classes

Both have the same set of grapheme natural classes.

Environments

Both have the same set of environments.

Sonority Hierarchy

Both have the same set of natural classes in the sonority hierarchy.

Syllabification Parameters

Codas allowed is the same.

Onset maximization is the same.

Onset principle differs.

First	Second
Onsets are not required	All but the first syllable must have an onset

Words

The following words differ between the two.

First	Second	Syllabification
ababrastro		a.bab.ra.stro
	ababrastro	a.bab.ra.stro
abida		a.bi.da
	abida	a.bi.da

Compare | Cancel | Help

13.4.4 Moraic approach comparison

When I did this for one project while showing the Moraic approach, I got what is shown in example (61).

(61)

Compare Implementations — □ ✕

First implementation:

Current implementation   Browse

◉ Current implementation   ○ Choose a backup

Second implementation:

C:\Users\Andy Black\Documents\My Asheninka\Moraic\EnglishIPA\Backups\CVTestData20210126-15114   Browse

Moraic Approach Comparison between:

1. Current implementation
2. C:/Users/Andy Black/Documents/My Asheninka/Moraic/EnglishIPA/Backups/CVTestData20210126-151141.ashebackup (Testing)

Performed on February 2, 2021 2:23 PM

Segment Inventory

Both have the same segment inventory.

Grapheme Natural Classes

Both have the same set of grapheme natural classes.

Environments

Both have the same set of environments.

Sonority Hierarchy

Both have the same set of natural classes in the sonority hierarchy.

Syllabification Parameters

Codas allowed is the same.

Onset maximization is the same.

Onset principle differs.

First	Second
Onsets are not required	Every syllable must have an onset

Weight by position differs.

First	Second
Yes	No

Maximum moras per syllable differs.

| First	Second
1	2

Compare   Cancel   Help

13.4.5 Nuclear projection approach comparison

When I did this for one project while showing the Nuclear projection approach, I got what is shown in example (62).

(62)

Compare Implementations ▯ ▢ ✕

First implementation:

C:\Users\Andy Black\Documents\My Asheninka\NuclearProjection\EnglishIPA\Backups\English20210326-142148.a [Browse]

○ Current implementation ● Choose a backup

Second implementation:

C:\Users\Andy Black\Documents\My Asheninka\NuclearProjection\EnglishIPA\Backups\English20210317-135649.a [Browse]

NP Approach Comparison between:

1. C:/Users/Andy Black/Documents/My Asheninka/NuclearProjection/EnglishIPA/Backups/English20210326-142148.ashebackup (No Word final adjoin)
2. C:/Users/Andy Black/Documents/My Asheninka/NuclearProjection/EnglishIPA/Backups/English20210317-135649.ashebackup (After getting all but ʃɹɪl to parse correctly.)

Performed on June 24, 2021 11:35 AM

Segment Inventory

Both have the same segment inventory.

Grapheme Natural Classes

Both have the same set of grapheme natural classes.

Environments

Both have the same set of environments.

Sonority Hierarchy

Both have the same set of natural classes in the sonority hierarchy.

Syllabification Parameters

The onset principle is the same.

Natural Classes

Both have the same natural classes.

Filters

Both have the same filters.

The order of the filters is shown below.

First	Second
Disallow non-strident coronal before labial in onset {Fail} [Onset] ([Non-stridentCoronal] l)	Disallow non-strident coronal before labial in onset {Fail} [Onset] ([Non-stridentCoronal] l)
Disallow stop/nasal in onset {Fail} [Onset] ([Stop] [N])	Disallow stop/nasal in onset {Fail} [Onset] ([Stop] [N])
Disallow two labials in onset {Fail} [Onset] ([Labial] [Labial])	Disallow alveo-palatal before sonorant in onsets {Fail} [Onset] ([Alveo-palatal] [Sonorant])

[Compare] [Cancel] [Help]

13.4.6 Optimality Theory approach comparison

When I did this for one project while showing the Optimality Theory approach, I got what is shown in example (63).

(63)

Compare Implementations

First implementation:

| Current implementation | Browse |

○ Current implementation   ○ Choose a backup

Second implementation:

| C:\Users\Andy Black\Documents\My Asheninka\OT\CVTestData\Backups\CVTestData20210621 | Browse |

OT Approach Comparison between:

1. Current implementation
2. C:/Users/Andy Black/Documents/My Asheninka/OT/CVTestData/Backups/CVTestData20210621-154555.ashebackup (Base)

Performed on June 24, 2021 11:27 AM

Segment Inventory

Both have the same segment inventory.

Grapheme Natural Classes

Both have the same set of grapheme natural classes.

Environments

Both have the same set of environments.

Constraints

The following constraints differ between the two.

First	Second
*Margin/V [V] | o	*Margin/V [V] | {o, c}

Constraint rankings
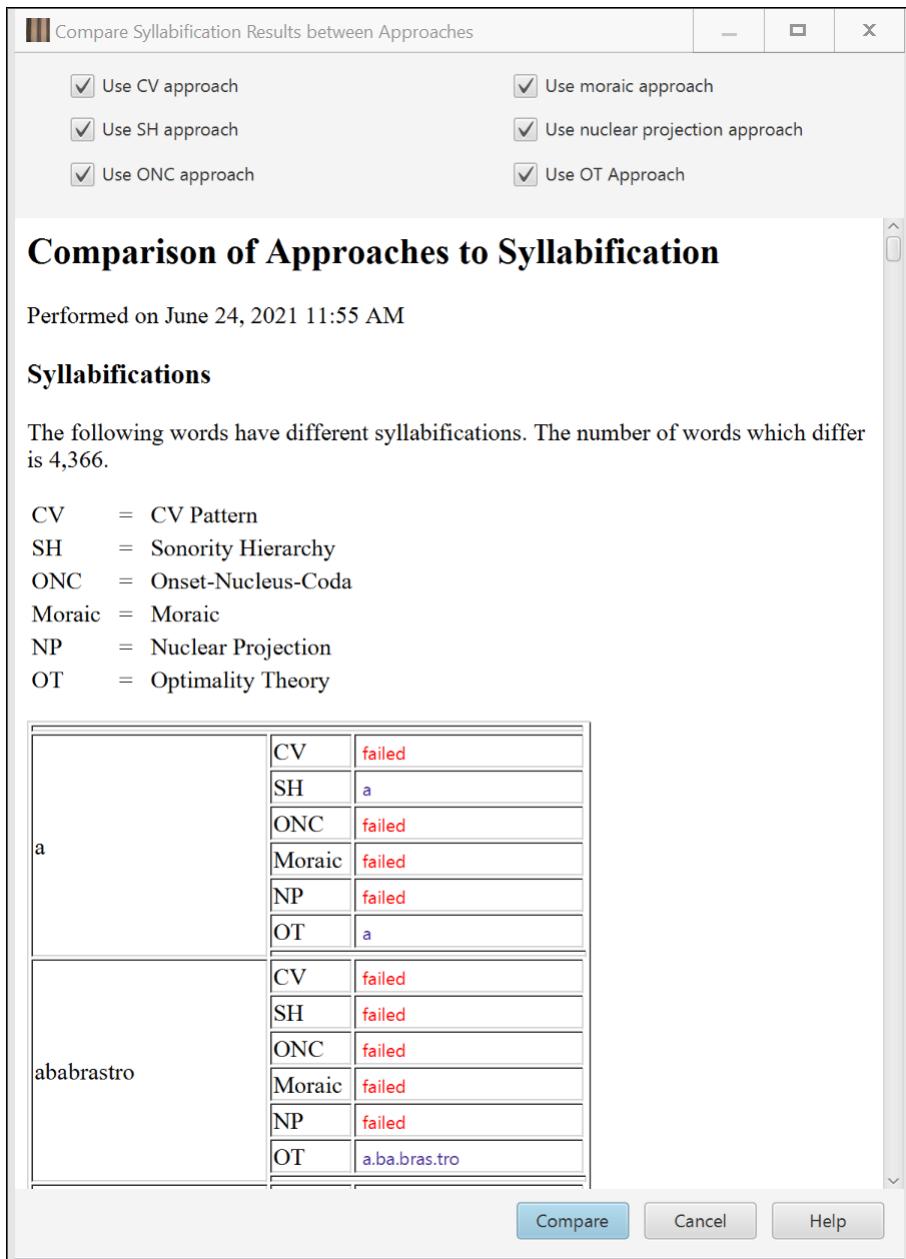
Both have the same constraint rankings.

The order of the constraint rankings is shown below.

| Compare | Cancel | Help |

13.5 Compare syllabification results between approaches

Asheninka also offers a way to more easily see how the words are syllabified between the approaches. Use the Tools / Compare Syllabification Results between Approaches menu item to do this. This does a comparison of the predicted syllabification of all words and shows the result in a dialog box. In one implementation I did, it looked like what is in example (64).

(64)

Compare Syllabification Results between Approaches	— □ X
☑ Use CV approach	☑ Use moraic approach
☑ Use SH approach	☑ Use nuclear projection approach
☑ Use ONC approach	☑ Use OT Approach

Comparison of Approaches to Syllabification

Performed on June 24, 2021 11:55 AM

Syllabifications

The following words have different syllabifications. The number of words which differ is 4,366.

CV	=	CV Pattern
SH	=	Sonority Hierarchy
ONC	=	Onset-Nucleus-Coda
Moraic	=	Moraic
NP	=	Nuclear Projection
OT	=	Optimality Theory

a	CV	failed
	SH	a
	ONC	failed
	Moraic	failed
	NP	failed
	OT	a
ababrastro	CV	failed
	SH	failed
	ONC	failed
	Moraic	failed
	NP	failed
	OT	a.ba.bras.tro

Compare   Cancel   Help

Note that you can choose which approaches to use in the comparison. In the example, all currently implemented approaches were chosen. The result shows only those words which had some difference between the parses. For each such difference, it shows each word with the result of how each chosen approach parsed it. In the case above, no vowel initial words were allowed in the CV pattern approach so

that's why there were so many failures. Scrolling down more in the dialog, there was the following portion, which better illustrates the differences:

(65)

matayos	CV	ma.ta.yos
	SH	ma.ta.yos
	ONC	failed
	Moraic	ma.ta.yos
	NP	failed
	OT	ma.ta.yos
matihmakak	CV	ma.tih.ma.kak
	SH	ma.ti.hma.kak
	ONC	failed
	Moraic	ma.tih.ma.kak
	NP	failed
	OT	ma.tih.ma.kak
matikchiwak	CV	ma.tik.chi.wak
	SH	ma.tik.chi.wak
	ONC	failed
	Moraic	failed
	NP	failed
	OT	ma.tik.chi.wak

Like several other dialog boxes in **Asheninka**, you can keep this dialog box open while doing other things.
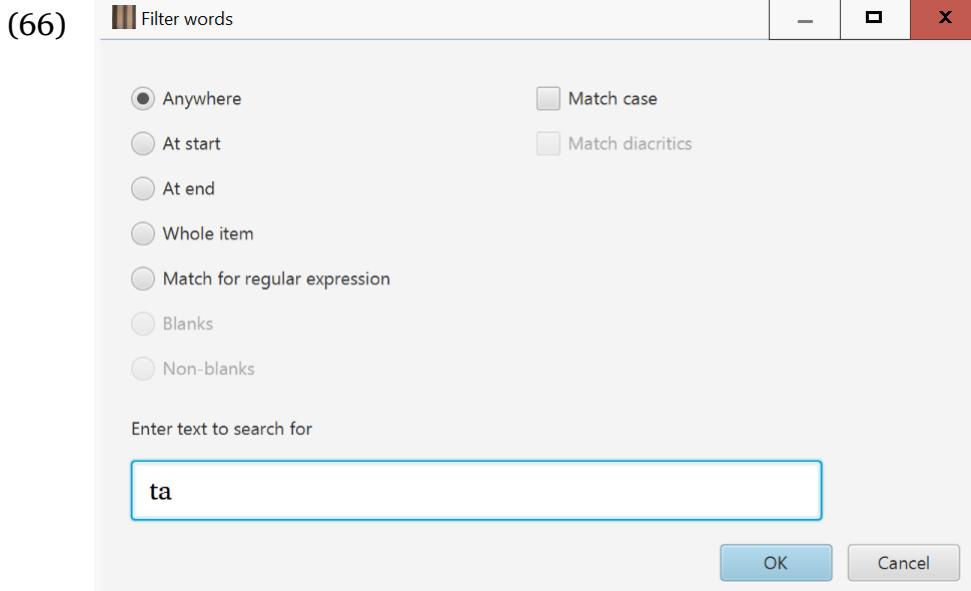
14 Filtering word columns

When using any of the Words views or the Predicted vs. correct views, **Asheninka** allows you to filter the columns containing words or syllabifications.

There are three ways to use one of these filters:

- Use Tools and then choose any of Filter words, Filter predicted syllable breaks, Filter correct syllable breaks or Remove all filters.
- Click on one of the filter-oriented toolbar buttons:
  - ⧨ for Filter words
  - ⧨ for Filter predicted syllable breaks
  - ⧨ for Filter correct syllable breaks
  - ⧨ for Remove all filters
- Right-click on the column header. This brings up a context menu containing the options available for that column. For the Words views, these have an additional option to remove the filter for just the column you right-clicked on.

A filter dialog looks like what is in example (66).

(66)

```
┌─────────────────────────────────────────────────────────────────────┐
│ ▌▌▌ Filter words                                        ─    □    ✕   │
├─────────────────────────────────────────────────────────────────────┤
│                                                                       │
│    ⦿ Anywhere                          ☐ Match case                   │
│                                                                       │
│    ○ At start                          ☐ Match diacritics             │
│                                                                       │
│    ○ At end                                                           │
│                                                                       │
│    ○ Whole item                                                       │
│                                                                       │
│    ○ Match for regular expression                                     │
│                                                                       │
│    ○ Blanks                                                           │
│                                                                       │
│    ○ Non-blanks                                                       │
│                                                                       │
│    Enter text to search for                                           │
│                                                                       │
│    ┌─────────────────────────────────────────────────────────────┐   │
│    │ ta                                                          │   │
│    └─────────────────────────────────────────────────────────────┘   │
│                                                                       │
│                                          OK          Cancel           │
└─────────────────────────────────────────────────────────────────────┘
```

You can choose to filter on the text you enter at the bottom using various methods:

1. The text occurs anywhere within the word or syllabification.
2. The text occurs at the start of the word or syllabification.
3. The text occurs at the end of the word or syllabification.
4. The text describes the entire word or syllabification.
5. The text is a regular expression which should match the word or syllabification.
6. Show only blanks.
7. Show only non-blanks.

The last two methods are not available for the Words column or the Predicted vs. correct column.[15] They are available for both the Predicted Syllable Breaks and the Correct Syllable Breaks columns.

In addition, you can tell it to match upper/lower case or not.

Regular expression formulas use what is documented at https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html beginning at *Summary of regular-expression constructs*. If the text you type cannot be parsed as a regular expression, then when you click on the OK button, an error dialog box will appear, showing where the error occurred and giving you an idea of what the problem is.

When creating a filter for any of the Predicted vs. correct views, **Asheninka** looks for matches for the shape of the word, not the shape of the syllabification (even though what is shown are syllabifications). That is, no periods are matched.

Whenever a filter is being applied, the column header will have a yellow background color. **Asheninka** will remember which filters have been used from previous sessions. It also keeps track of these filters on an approach by approach basis. That is, you can have different filters for each approach.

---

[15]The reason is that there are no blank items to show; what is there is all there is. So neither of these options make sense for these columns.

Appendix A. CV algorithm in detail

The main algorithm is as follows:

1. It does a left-to-right sweep of the word, trying to find the sequence of segments which covers the whole word. It uses the graphemes defined in the segment information to determine a match with a segment. It tries all graphemes which match at the current point it is looking at in the left-to-right sweep. It tries the longest match first, but also applies any environments associated with the grapheme. See section 4.1 for more on this. If it cannot find a sequence that covers the whole word, it reports an error of "Failure: could not parse into segments beginning at '*some characters*'" and quits. The '*some characters*' part indicates the place in the word where it could not find a character or character sequence that matched any graphemes in any of the segments. Most likely, either there was a typo in the word or a grapheme is missing from some segment or its environment was not met. You can use the **Try a Word** tool to get more details (see section 4.8).

2. If it succeeded in finding a sequence of segments, it then performs a left-to-right sweep of the segments it posited, trying to find a sequence of natural classes that covers the entire sequence of segments. It uses the set of segments or natural (sub-)classes defined within a natural class to determine a match with a natural class. If two or more natural classes match, it tries each of them in turn. If it cannot find a sequence of natural classes that covers the entire sequence of posited segments, it reports an error of "Failure: could not parse into natural classes; did find classes '*some classes*' covering graphemes '*some graphemes*'" and quits. The '*some classes*' part indicates the sequence of natural classes that were found and the '*some graphemes*' part indicates which graphemes were included in those natural classes. Hopefully this will help you figure out why other natural classes were not found for this word.

3. If it succeeded in finding a sequence of natural classes, it performs a left-to-right sweep of the sequence of the natural classes it posited, trying to find a sequence of syllable patterns that covers the entire sequence. It tries the syllable patterns in the order they are arranged in the **CV Syllable Patterns** view (see section 4.3 below). If it cannot find a sequence of syllable patterns that covers the entire sequence of posited natural classes, it reports an error of "Failure: could not parse natural classes into syllables" and quits.

4. If it succeeded in finding a sequence of syllable patterns, it outputs the syllabification into the **Predicted Syllable Breaks** field and reports "Success".

Appendix B. Sonority hierarchy algorithm in detail

The main algorithm is as follows:

1. It does a left-to-right sweep of the word, trying to find the sequence of segments which covers the whole word. It uses the graphemes defined in the segment information to determine a match with a segment. It tries all graphemes which match at the current point it is looking at in the left-to-right sweep. It tries the longest match first, but also applies any environments associated with the grapheme. See section 4.1 for more on this. If it cannot find a sequence that covers the whole word, it reports an error of "Failure: could not parse into segments beginning at '*some characters*"" and quits. The '*some characters*' part indicates the place in the word where it could not find a character or character sequence that matched any graphemes in any of the segments. Most likely, either there was a typo in the word or a grapheme is missing from some segment or its environment was not met.[16] You can use the **Try a Word** tool to get more details (see section 5.7).

2. If it succeeded in finding a sequence of segments, it then performs a left-to-right sweep of the segments it posited. It places the first segment of the word in the first syllable. It then tries to find the natural class of the current segment and the natural class of the following segment (unless the current segment is at the end of the word and also not already in a syllable; in this case, it adds the current segment to the current syllable). It then compares the "level" in the sonority hierarchy of these two natural classes and processes them as follows:

    a. If they are at the same level, a new syllable is created and the second segment is added to this new syllable.

    b. If the natural class of the current segment is less sonorous than the natural class of the following segment, then it adds the following segment to the current syllable.

    c. If the natural class of the current segment is more sonorous than the natural class of the following segment, then it looks at the segment after the following segment. There are two cases considered:

        i. If the second segment's natural class is the same or more sonorous than the third segment's natural class, than the second segment is added to the current syllable. A new syllable is created and the third segment is added to it.

        ii. Otherwise, a new syllable is created and the second segment is added to it.

3. Whenever a segment is not found in any natural class, an error is reported.

4. If it succeeded in finding natural classes for all segments, it outputs the syllabification into the **Predicted Syllable Breaks** field and reports "Success".

---

[16]This is the exact same algorithm used by the CV pattern approach.

Appendix C. ONC algorithm in detail

The main algorithm for the Onset-nucleus-coda approach is as follows:

1. It does a left-to-right sweep of the word, trying to find the sequence of segments which covers the whole word. It uses the graphemes defined in the segment information to determine a match with a segment. It tries all graphemes which match at the current point it is looking at in the left-to-right sweep. It tries the longest match first, but also applies any environments associated with the grapheme. See section 4.1 for more on this. If it cannot find a sequence that covers the whole word, it reports an error of "Failure: could not parse into segments beginning at '*some characters*'" and quits. The '*some characters*' part indicates the place in the word where it could not find a character or character sequence that matched any graphemes in any of the segments. Most likely, either there was a typo in the word or a grapheme is missing from some segment or its environment was not met.[17] You can use the **Try a Word** tool to get more details (see section 6.11).

2. If it succeeded in finding a sequence of segments, it then looks at the first segment. If it is not an onset and the Onset Principle is set to *Every Has Onset*, then it reports an error of "Onsets are required but segment 1 was not an onset." and quits.

3. Otherwise, it performs a left-to-right sweep of the segments. For each segment, it compares the sonority of this onset to the sonority of the following segment. What happens next depends on the onset, nucleus, or coda status of the segment.

4. It looks at the classification of the segment (onset, nucleus, and/or coda).

   a. If it can be an onset, it compares the sonority of this onset to the sonority of the following segment. If the following segment's sonority is less than or equal to this segment's sonority, this segment is added to the current syllable.

   b. Otherwise, it compares the sonority of the segment to the sonority of the following segment. It also assigns a type (or state) to the segment's position, depending on what the last segment was. The types are:
      i. Onset
      ii. Onset or nucleus
      iii. Nucleus
      iv. Nucleus or coda
      v. Coda
      vi. Coda or onset
      vii. Unknown
      The initial type is set to unknown.

   c. Here is what it does for each type:
      i. Onset:

---

[17]This is the exact same algorithm used by the CV pattern approach.

1. If the segment can be an onset and its sonority is less than the sonority of the following segment, it adds the segment to the syllable as an onset. The type is set to be an onset or a nucleus.
2. Otherwise, it sets the type to be a nucleus.

ii. Onset or nucleus:

1. If the segment can be an onset and its sonority is less than the sonority of the following segment, it adds the segment to the syllable as an onset. The type is set to be an onset or a nucleus.
2. Otherwise, if the segment can be a nucleus, it adds it to the syllable as a nucleus. The type is set to be a nucleus or a coda.
3. Otherwise, the parsing quits as a failure.

iii. Nucleus:

1. If the segment can be a nucleus, it is added to the syllable as a nucleus. The type is set to be a nucleus or a coda.
2. Otherwise, the parsing quits as a failure.

iv. Nucleus or coda:

1. If the segment can be a nucleus, it is added to the syllable as a nucleus. The type is set to be a nucleus or a coda.
2. Otherwise, if the segment can be a coda and the Codas Allowed parameter is set, then
   a. If the segment can also be an onset and it is less sonorous than the following segment:
      i. If the Onset Maximization parameter is set, the syllable is added to the word and new a syllable is created. The type is set to onset or nucleus if the Onset Principle is set to *Onsets Not Required*; otherwise the type is set to onset.
      ii. Otherwise, if the following segment is not an onset and the Onset Principle is not set to *Onsets Not Required,* the syllable is added to the word and new a syllable is created. The type is set to onset or nucleus.
      iii. Otherwise, the segment is added to the syllable as a coda and a new syllable is started. The type is set to onset or nucleus if the Onset Principle is set to *Onsets Not Required*; otherwise it is set to onset.
   b. Otherwise:
      i. The segment is added to the syllable as a coda and a new syllable is started. The type is set to onset or nucleus if the Onset Principle is set to *Onsets Not Required*; otherwise the type is set to onset.
3. Otherwise, the syllable is added to the word, a new syllable is created and the type is set to be an onset.

v. Coda or onset:

1. If the segment can be a coda, the Codas Allowed parameter is set, and the segment is more sonorous than the following segment, it adds the segment to the syllable as a coda. The type is set to be a coda or an onset.

    2. Otherwise, the segment is added to the syllable as a coda and a new syllable is started. The type is set to onset or nucleus if the Onset Principle is set to *Onsets Not Required*; otherwise the type is set to onset.

      vi. Coda: (This never happens actually.)

5. If it succeeded in processing all the segments in the word, it outputs the syllabification into the **Predicted Syllable Breaks** field and reports "Success".

If you have defined any filters or templates, then these also are applied during this process. In particular, filters are only applied for onset, nucleus, coda, and rime constituents.

Filters come in two varieties: fail and repair. When a filter is matched and is marked as fail, then the syllable parsing stops at that point and seeks other possibilities, if any. When the filter matches and is marked as repair, it will seek to fix up the syllable constituents around it to fix the situation. For example, for English, a word such as *Atlantic* (ætlæntɪk) will normally syllabify as æ.tlæn.tɪk but it needs to syllabify as æt.læn.tɪk. By adding an onset repair filter that matches the tl sequence, one can obtain the correct result.

Templates can be applied to onset, nucleus, coda, word initial, and word final constituents. For nucleus, the set of slots must match. For the others, the set of slots can override the **Sonority Hierarchy**.

Appendix D. Moraic algorithm in detail

The main algorithm for the Moraic approach is as follows:

1. It does a left-to-right sweep of the word, trying to find the sequence of segments which covers the whole word. It uses the graphemes defined in the segment information to determine a match with a segment. It tries all graphemes which match at the current point it is looking at in the left-to-right sweep. It tries the longest match first, but also applies any environments associated with the grapheme. See section 4.1 for more on this. If it cannot find a sequence that covers the whole word, it reports an error of "Failure: could not parse into segments beginning at '*some characters*'" and quits. The '*some characters*' part indicates the place in the word where it could not find a character or character sequence that matched any graphemes in any of the segments. Most likely, either there was a typo in the word or a grapheme is missing from some segment or its environment was not met.[18] You can use the **Try a Word** tool to get more details (see section 7.11).

2. If it succeeded in finding a sequence of segments, it then processes the segments one after the other. It starts in an "Onset" state.

3. When in the "Onset" state,

_____

[18]This is the exact same algorithm used by the CV pattern approach.

a. If there is an onset template, it sees if the sequence of segments matches it. If it does, it tries to add the onset segments to the syllable and assumes that any following segments must be mora-bearing.

b. If no onset templates exist or match,

   i. if the segment is not mora-bearing, then:

      1. If there are syllable templates

         a. and one applies, it adds the segment as an onset;

         b. if none apply, syllabification fails.

      2. If the sonority of this segment is less than what follows, if adds the segment as an onset;

      3. otherwise, it fails to syllabify.

   ii. Otherwise,

      1. if onsets are required at this point in the parse of the word, syllabification fails.

      2. Otherwise, it will see if the segment will work as a mora-bearing segment after checking for sonority issues. That is, it changes the state to "Mora."

4. When in the "Mora" state,

   a. if the segment bears one or more moras,

      i. If the syllable can have another mora,

         1. if no syllable templates apply and there is at least one mora in the syllable already, finish the current syllable and start a new one. The state is changed to "Onset."

         2. Otherwise, add the segment to the syllable as a mora-bearing segment.

      ii. If the syllable cannot have another mora (because the maximum number of moras in a syllable has been reached), then finish this syllable and start a new one. The state is changed to "Onset."

   b. If the segment can be a coda,

      i. if there already is a coda segment and an onset template applies, it finishes the current syllable, creates a new one, and adds the segments matching the onset template to the new syllable. The state is changed to "Mora."

      ii. If weight-by-position is in effect, and another mora can be added to the syllable, it adds the segment as mora-bearing.

      iii. Otherwise it adds the segment to the preceding mora in the syllable.

   c. Otherwise,

      i. if the syllable has at least one mora it adds the syllable to the word and starts a new syllable. The state is changed to "Onset."

         1. If any onset templates apply, it adds the matching segments to the syllable and the state is changed to "Mora."

         2. Otherwise, if any word final templates apply, syllabification succeeds.

      ii. Otherwise, syllabification fails.

5. When all of the segments have been processed, the current syllable is added to the word, if needed.

Whenever a segment is added to a syllable as an onset, it also tries to apply any onset repair filters and applies any onset fail filters.

If it succeeded in processing all the segments in the word, it outputs the syllabification into the **Predicted Syllable Breaks** field and reports "Success".

Appendix E. Nuclear Projection algorithm in detail

The main algorithm for the Nuclear projection approach is as follows:

1. It does a left-to-right sweep of the word, trying to find the sequence of segments which covers the whole word. It uses the graphemes defined in the segment information to determine a match with a segment. It tries all graphemes which match at the current point it is looking at in the left-to-right sweep. It tries the longest match first, but also applies any environments associated with the grapheme. See section 4.1 for more on this. If it cannot find a sequence that covers the whole word, it reports an error of "Failure: could not parse into segments beginning at '*some characters*'" and quits. The '*some characters*' part indicates the place in the word where it could not find a character or character sequence that matched any graphemes in any of the segments. Most likely, either there was a typo in the word or a grapheme is missing from some segment or its environment was not met.[19] You can use the **Try a Word** tool to get more details (see section 8.11).

2. For each rule:

   a. Determine the set of segments, if any, that match the affected/context/level part of the rule:

      i. If it is a `Build` rule, match any affected segments.

      ii. If it is a `Attach` or `Augment` rule and the level is `N''`, match the affected immediately before the context.

      iii. If it is a `Attach` or `Augment` rule and the level is not `N''`, match the affected immediately after the context.

      iv. If it is a `Left adjoin` rule, match the affected immediately before the context.

      v. If it is a `Right adjoin` rule, match the affected immediately after the context.

   b. For each match:

      i. If the rule must obey the SSP, check the sonority between the affected and context segments. If the affected comes before the context, then sonority must be rising. If the affected comes after the context, then the sonority must decrease.

      ii. If sonority passes (or is ignored), then:

---

[19]This is the exact same algorithm used by the CV pattern approach.

    1. If it is a `Build` rule, create a new syllable and the constituent structure (syllable to `N''` to `N'` to `N` to the affected segment that matched).

    2. If it is an `Attach` or `Augment` rule, apply any appropriate filters for the level (`N''` is onset and `N'` is rime and coda). If any filter fails, quit. Otherwise add the affected segment to the constituent structure at the rule's level.

    3. If it is an `Left adjoin` or `Right adjoin` rule, adjoin the segment to a new instance of the level node.

3. Check every segment in the word to see if it has been made a part of a syllable. If not, fail.

4. Check the Onset Principle setting:

    a. If the *All But First Has Onset* is set, then if any syllable after the first does not have an onset (i.e., a segment in the `N''` level), fail.

    b. If the *Every Has Onset* is set, then if any syllable does not have an onset (i.e., a segment in the `N''` level), fail.

    c. If the *Onsets Not Required* is set, then there is nothing to do.

5. If it succeeded in processing all the segments in the word, it outputs the syllabification into the **Predicted Syllable Breaks** field and reports "Success".

Appendix F. OT algorithm in detail

The main algorithm for the Optimality Theory approach is as follows:

1. It does a left-to-right sweep of the word, trying to find the sequence of segments which covers the whole word. It uses the graphemes defined in the segment information to determine a match with a segment. It tries all graphemes which match at the current point it is looking at in the left-to-right sweep. It tries the longest match first, but also applies any environments associated with the grapheme. See section 4.1 for more on this. If it cannot find a sequence that covers the whole word, it reports an error of "Failure: could not parse into segments beginning at '*some characters*'" and quits. The '*some characters*' part indicates the place in the word where it could not find a character or character sequence that matched any graphemes in any of the segments. Most likely, either there was a typo in the word or a grapheme is missing from some segment or its environment was not met.[20] You can use the **Try a Word** tool to get more details (see section 9.9).

2. Initialize the candidate grid.

3. Apply word initial/word final housekeeping: remove the coda structural option from the first segment in the word and the onset structural option from the last segment of the word.

4. For each constraint in the first ranking in the OT Constraint Rankings:

---

[20]This is the exact same algorithm used by the CV pattern approach.

a. Determine the set of segments, if any, that match the affected element(s) part of the constraint.

b. For each match:

    i. If the set of structural options in the constraint are exactly the same as the remaining structural options in the segment, do nothing.[21]

    ii. Otherwise, remove any structural options in that segment per the structural options of the constraint as long as there is at least one structural option left.

c. If the "onset before coda" housekeeping applies, remove the onset.

d. If every segment has at most one structural option in it, check no more constraints.

5. Check every segment in the word to see if it has been made a part of a syllable and has at most one structural option. If not, fail.

6. If it succeeded in processing all the segments in the word, it outputs the syllabification into the **Predicted Syllable Breaks** field and reports "Success".

Appendix G. Known issues

The following are some issues we are aware of that have yet to be fixed.

1. The "Segment or Natural Class" field in CV Natural Classes may show a class closing square bracket at the beginning of the field when the vernacular language is set to show right-to-left. Similarly, the "Grapheme or Natural Class" field will do the same.

2. The dialog boxes which can remain open while you do other things do not have a way to be minimized.

3. If you try to import a word list that is not in UTF-8 encoding, then you will get a "A serious error happened." dialog box. Convert your file to UTF-8 and try again.

4. On macOS: if you double click an **Asheninka** description file in **Finder**, then **Asheninka** will open the last opened file, not the one you clicked on. This is due to a communication problem between the macOS operating system and the Java language **Asheninka** is written in.

5. When you move a **Asheninka** file from one computer to another, you may well need to reset the keyboards manually. This is because the keyboard settings may be different on the two machines.

---

[21] This is because removing all that match would leave no structural options at all. This would be an invalid state.

Appendix H. Sample data sets

The **Asheninka** website has some sample data sets in case you find these of value. There are two. They are located at http://software.sil.org/downloads/r/asheninka/resources/SampleProjects.zip

H.1 English IPA

This has English words written in IPA (IPA 2016) per my dialect. It illustrates the need for some templates and filters which are described in the "Introduction to Syllabification" document. You can read this document by clicking here or by using the Help / Introduction to syllabification menu item.

Please note that the CV pattern approach fails to parse many of the words as does the Sonority hierarchy approach. The other approaches have successful parses for the words.

H.2 Asheninka Pajonal

This has an implementation for the Pajonal variety of Asheninka (ISO-639-3 code cjo). Here are some observations about it:

1. The basic syllable pattern is (C)V(V)(N).
2. Onsets are required for all syllables but the first.
3. The nasals in the coda are homorganic with the following onset. We do not actually check for this.
4. The second vowel of the nucleus is either geminate or an e which forms a diphthong with a preceding a or o.
5. There is one template used only for the Moraic approach. No filters are needed.

In this implementation, I chose to model the geminates and diphthongs as separate segments. The reasons for this are as follows:

1. Fewer CV syllable patterns are needed for the CV pattern approach.
2. The Sonority hierarchy approach forces a syllable break whenever there are two consecutive top-level segments. Since vowels are the highest level, this means the geminates and diphthongs are broken into two syllables. This would be incorrect.

All words parse except for an onomatopoeiaic word, with the exception of the Sonority hierarchy approach. This word parses for that approach.

Appendix I. Why is it called Asheninka?

This syllable parsing program is called **Asheninka** for historic reasons. (The word is pronounced a.ˈʃɛ.nɪŋ.ka.)

In late 1983 my family and I were living in the jungles of Peru and David Payne came and asked me if I would create a **Consistent Changes**[22] table for him that would insert discretionary hyphens in Asheninka text. The algorithm he suggested was the CV Patterns approach (see section 4). Asheninka has long words and typesetting material in that language would improve readability with such a table. In 1984 I wrote such a table.

While it was functional, it ran slowly. I then wrote the **Hyphen** program (Black et al. 1987) to improve the efficiency. It implemented the same basic approach. Amazingly, the **Hyphen** program is still being used (albeit occasionally) today.

Because of this beginning, I chose to call this tool **Asheninka**. The program icon (shown in example (67) below) is the kind of material used for clothing by Asheninka people.[23]

(67)

References

Black, H. Andrew, Fred Kuhl, Kathy Kuhl, and David J. Weber. 1987. *Document preparation aids for non-major languages.* Occasional publications in academic computing Issue 7. Dallas, TX: Summer Institute of Linguistics. http://www.sil.org/resources/publications/entry/29601 (accessed 24 February 2017)

Hammond, Michael. 1997. Parsing in OT. University of Arizona. Manuscript. http://roa.rutgers.edu/files/222-1097/222-1097-HAMMOND-0-0.PDF (accessed 11 December 2020)

IPA. 2016. International Phonetic Association. https://www.internationalphoneticassociation.org/ (accessed 9 November 2016)

SIL International. 2005. Consistent Changes Program. http://scripts.sil.org/cms/scripts/page.php?site_id = nrsi&id = cc-grkuni (accessed 24 February 2017)

---

[22]A more current version of this program is SIL International (2005).

[23]This image was gratefully taken from here on 12 November, 2015.

Index