1 Nuclear Projection approach

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

1. segments (1.1)
2. a sonority hierarchy via natural classes (1.2)
3. syllabification parameters (1.3)
4. CV Natural Classes 1.4)
5. NP rules (1.5)
6. grapheme natural classes (1.9)
7. environments (1.10)

If needed to successfully parse your language, you may also need to define NP filters 1.6).

After that, you either use a existing word list, import a word list (see) 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 1.7 and 1.11 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 for details.

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

1.1 NP segment inventory

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

1.2 Sonority hierarchy

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

1.3 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 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.

1.4 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 for details.

1.5 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 and is usually built on vowels. The other rules tend to 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 (1).

(1)

Action	Description
`Attach`	Attach the affected segment at the node level of the rule. This rule is applied only once per syllable.
`Augment`	Augment the affected segment at the node level of the rule. This rule can be applied more than once per syllable.
`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 (2).

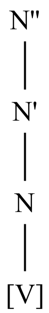
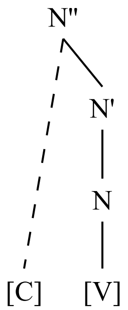
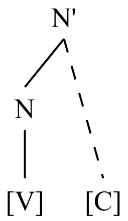
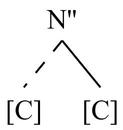
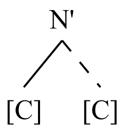
(2)

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 (3) below lists the standard five rules.

(3)

Rule	**Affected**	**Context**	**Action**	**Level**	**Diagram**
Nucleus	[V]		`Build`	`All`	
Onset	[C]	[V]	`Attach`	`N''`	
Coda	[C]	[V]	`Attach`	`N'`	
Augmented onsets	[C]	[C]	`Augment`	`N''`	
Augmented codas	[C]	[C]	`Augment`	`N'`	

Example (4) has some examples from an English project. They are all repair filters. Example (5) has an example from Quiegolani Zapotec which makes use of the context in the preceding syllable ([Obs] is a class of obstruents).

(4)

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.

(5)

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.

1.6 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 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 fllter as in example (6). 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.

(6) [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 (7)

(7) *[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 maɪʃlænd, the context is met and so the filter does block adding the ʃ as desired.

1.7 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 for more on this view.

1.8 Predicted vs. correct Moraic 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.

1.9 Grapheme natural classes

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

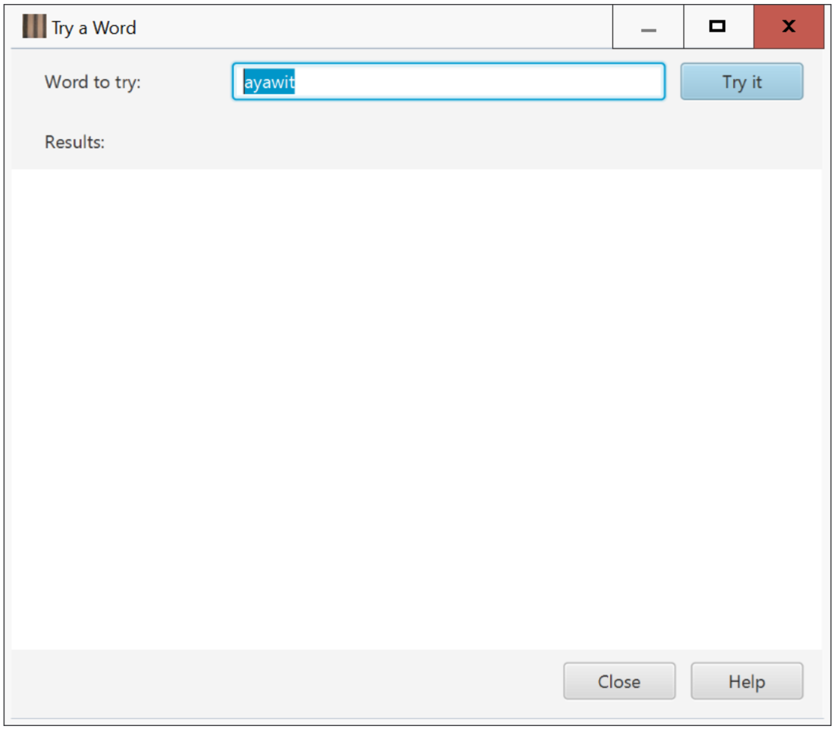
1.10 Environments

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

1.11 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 (). 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 for the CV pattern approach, Sonority hierarchy approach, the Onset-nucleus-coda approach, and the one for the Moraic approach can be set independently; you can have several **Try a Word** dialog boxes open at the same time if you wish.

(8)



Word to try: ayawit

Results:

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).

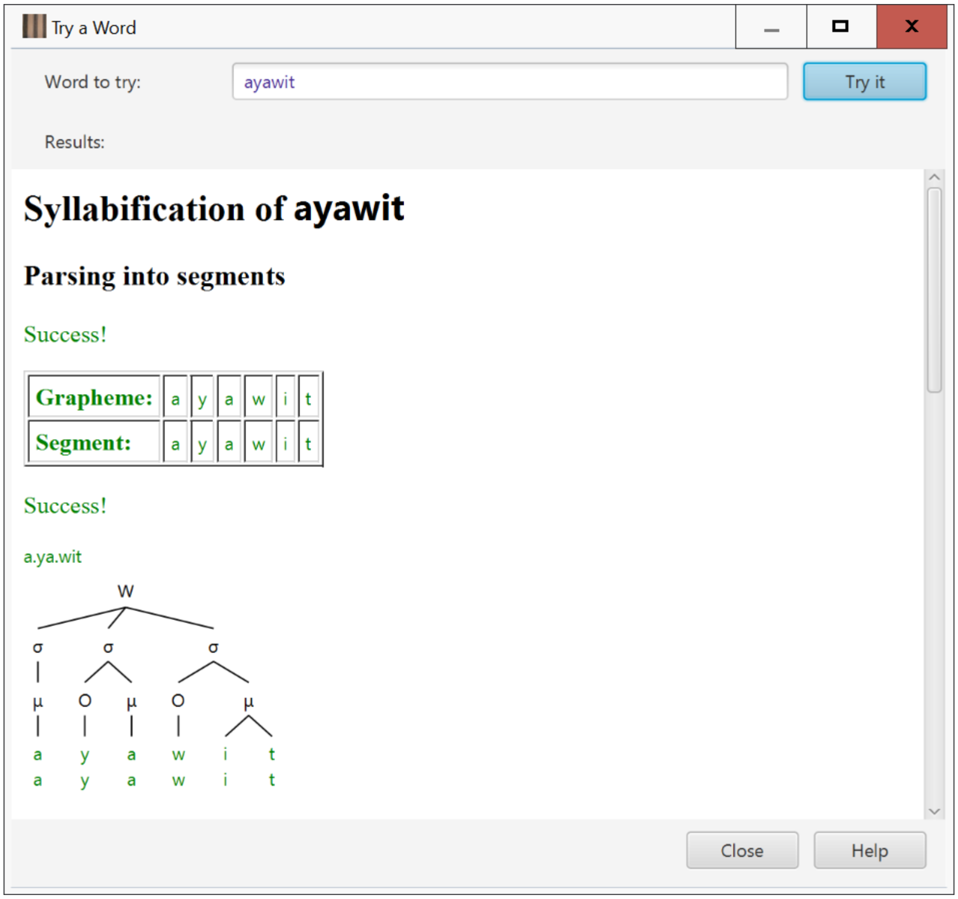
When you either press the **Enter** key or click on the **Try it** button, **Asheninka** will try to parse the word in the text box and report the result in the Results portion of the dialog box.

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 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 ().

(9)



Syllabification of ayawit

Parsing into segments

Success!

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

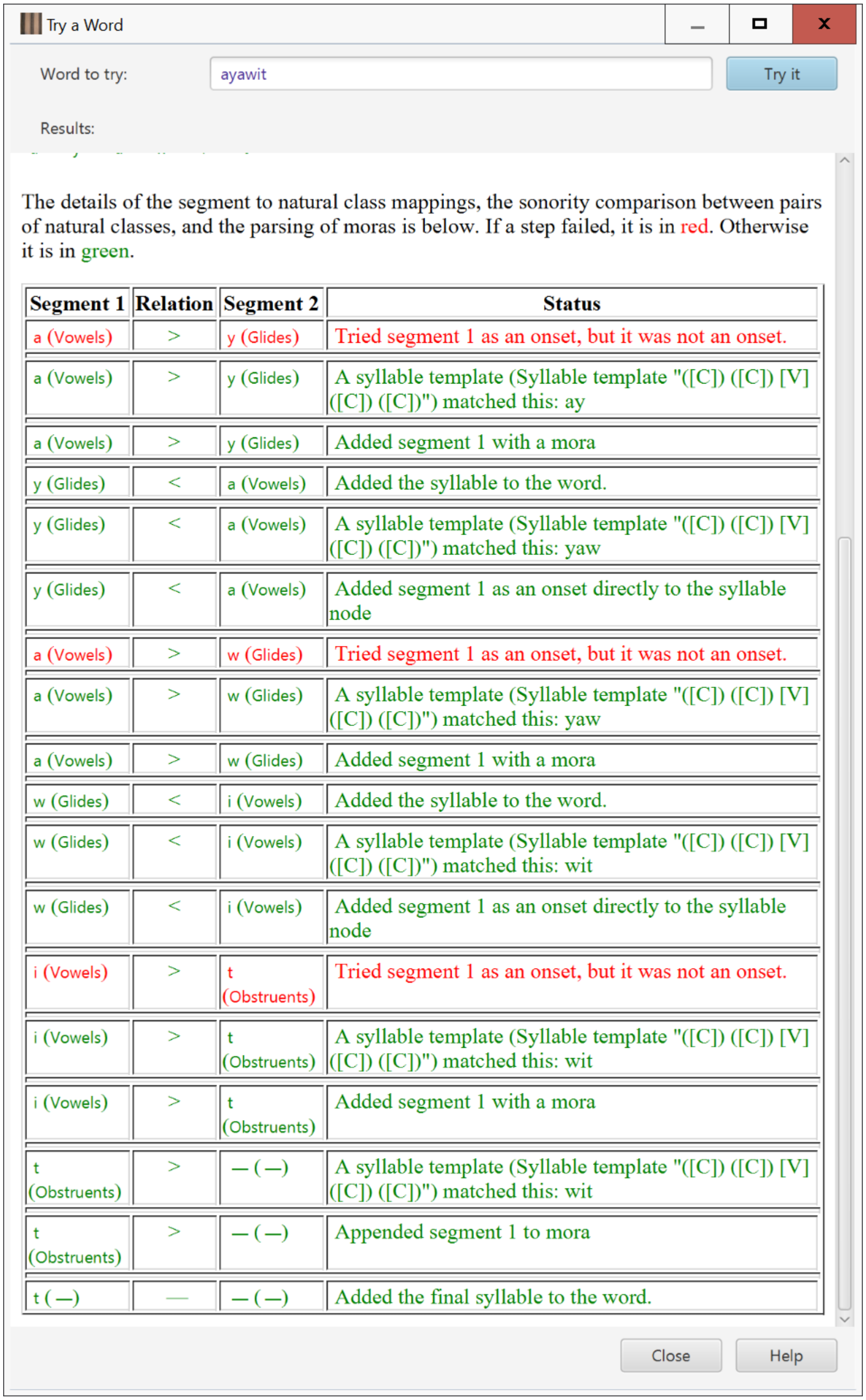
Success!

a.ya.wit

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 O 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:

(10)



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.

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 ().

(11)

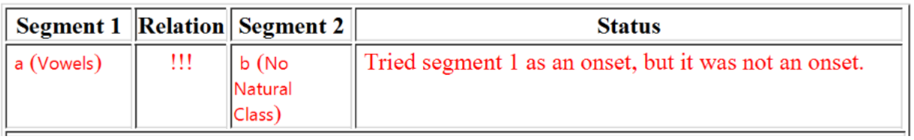
Relation	Meaning
<	Segment 1 is less sonorous than segment 2.
=	Segment 1 is equal in sonority to segment 2.
>	Segment 1 is more sonorous than segment 2.
!!!	The segment was not in any natural class.

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 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 ().

(12)



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 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

2