How to use the Morphoton

Raphael Finkel raphael@cs.uky.edu Daniel Kaufman dkaufman@qc.cuny.edu

December 13, 2024

1 Introduction

The Morphoton is a web-accessible facility that applies user-supplied, language-dependent phonological and morphological rules to roots and morphological property sets to automatically derive surface forms for those roots. It is intended both as an aid in teaching morphology and to help researchers formalize morphological theories.

One can use the Morphoton in several ways.

1. Enter a morphological theory and have the Morphoton compute its results.
2. Choose a prepackaged morphological theory from the **Library**.
3. Click symbols in the **IPA chart** or the **Segments and features** section to see their phonological features.

The Morphoton has an **Instructions** section that summarizes usage. The current document is meant to be a more verbose introduction.

2 A sample theory from the library: Latin

To clarify how the Morphoton computes results of a morphological theory, we present a simple (and very incomplete) Latin theory. Use the top navigation bar to select **Library**. Click the Latin button. The browser submits this request to the server, which runs the Morphoton program. The Morphoton loads the

stored theory into the inputs sections (**Metadata**, **Preamble**, **Input values**, and **Rule blocks**). It then executes its instructions, filling in all the inputs and then showing the result in the **Latin Results** section.

The **Preamble** section specifies that the `Morphology` has two morphosyntactic properties; the morphosyntactic property (**MP**) `Tense` has two possible **values**: `Present` and `Future`. The **Input values** section specifies that that the desired morphosyntactic property set (**MPS**) is `1sg Present`.

2.1 Results

The **Latin Results** section shows a chart headed by the MPS. The first column in the chart lists the names of all the applicable rule blocks. Each subsequent column shows a particular root, such as `love:am`, then the effect of each applied rule on that form, finally arriving at the surface form, such as `amo`. Any rule that does not modify the form presents an empty cell. We'll just call all these forms, whether root, intermediate, and surface, simply **forms**.

The results chart has a few features.

- If the chart is too wide for the browser, a scrollbar appears at the bottom that you can use to scroll horizontally.
- If you hover over a cell, a tooltip appears naming the rule that it demonstrates.
- If you click a surface form, the Morphoton attempts to pronounce it.

The **Input values** line provides some interactive buttons. Click Create options to build pull-down menus so you can select the MP values you wish to place in the MPS. After you select the values, click Submit to see the resulting chart. The All button generates the pseudo-MPS `all`. When you submit that specification, the Morphoton generates many charts. You can reduce these charts to the `surface form` lines by clicking hide/show details near the top. Click it again to reveal the full charts.

2.2 Entering a theory

There are three major input sections.

- **Metadata**: optional extra information about the theory. The Metadata contains these fields:
  - Language, such as `Latin`

  - Date, such as `2023-09-01`
  - Language family, such as `Romance`
  - Code (Glottocode), such as `lati1261`
  - Sources, such as `Ørberg (2006)`
  - Implementation, such as `Raphael Finkel`
  - Details, such as `Verbs (pres/future, conj 1/2/4)`
  - Complexity, on a scale from 1 to 5, such `2`.

  The examples in the library follow these conventions.

- **Preamble**: This section is organized into blank-line separated specification regions.
  - The optional `Phonology` region can contain phonological modifications to the features discussed by Hayes Hayes (2009). In the case of the Latin theory, in order to use an IPA representation, we distinguish long and short vowels (such as, `ō` and `o`) by their `tense` property, using the IPA `o` and `ɔ`, respectively. The only exception is `ā` versus `a`, which we represent by `A` and `a`, respectively. To make that representation work, the Phonology region redefines `a`, which usually has `[0tense]`, to have `[-tense]`, and it introduces a custom phoneme `A` to be equivalent to `a` but to have the property `[+tense]`. We present more details in **??**.
  - The `Morphology` region contains a list of the MPs, such as `Tense` and `PersonNumber`, each on a separate line. For each MP, after its name and a colon, there is a list of its possible values. For Latin, the MP `Tense` can be either `Present` or `Future`.
  - The `Roots` region contains the lexicon to be analyzed, with each lexeme on a separate line. For each lexeme, after its gloss and a colon, there is a multi-tier specification of its form. For Latin, we use the tier `«cat»` to indicate the inflection class and the tier `«seg»` to indicate the stem.
  - The Preamble may also contain optional regions called `DisallowMPS`, `Results`, and `Template`. We discuss these regions later.

- **Rules**: This section is organized into blank-line separated rule blocks. Each block is prefaced by a single line indicating its **block name**, followed by an optional colon. The content of each block is a list of rewrite rules that convert forms. Ordinary blocks, such as the Latin `Theme` and `Tense` blocks, are **selective**, that is, the Morphoton selects at most one rule to apply. A block such as `Sandhi`, which has `(all)` after its name, is not selective; every rule is applied.

  A rule may have any number of **components**. For instance, the second rule in the `PersonNumber` block is

  ```
  «cat» {conj4} «morph» {1sg} {Future} «seg» 0 –> m / _>
  ```

This rule has three components, each prefaced by a tier name. Some components are **conditions**, such as the «cat» and «morph» components. Others are **changes**, such as the «seg» component. This rule only applies if all the conditions are satisfied, in particular, if the «cat» tier contains conj4 and the «morph» tier, which implicitly contains the selected MPS, has both 1sg and Future. If the conditions are satisfied, then all change components are attempted. Here, the **target**, which is the empty string represented by 0, becomes the **replacement** m in the context _>, which means that the target is followed immediately by the end of the word. Details of change components are in section 3.2.

In a selective block, the Morphoton selects a single rule by inspecting the conditions. Only rules whose conditions are satisfied are considered **available**. If multiple rules are available, the one with the most restrictive conditions is chosen, in accordance with the rules of Pāṇini precedence. The PersonNumber block has two rules that could apply if the MPS is 1sg Future and the category is conj4:

```
«morph» {1sg} «seg» 0 –> o / _>
«cat» {conj4} «morph» {1sg} {Future} «seg» 0 –> m / _>
```

The Morphoton selects the second rule, because it is more restrictive.

The Morphoton presents several buttons to assist in entering a theory.

- The Morphoton understands many non-ASCII symbols: IPA characters such as ɮ, characters with special meaning in change rules such as ⁴, and diacritics such as palatal, represented by ʲ. The keyboard buttons (separate for **Preamble** and **Rule blocks**) bring up a pane containing these symbols; clicking a symbol in that pane places that symbol at the current location in the associated input area.
- Submit The browser submits the current theory to the Morphoton without leaving the current page, so the user can use local editing commands such as <ctrl-Z> to return to a previous version of the theory. The Morphoton logs submissions to assist the developers.
- Refresh-submit The browser submits the current theory to the Morphoton and builds a new page with the result.
- Download The Morphoton downloads the theory in a text format that can be saved and later uploaded.
- Browse The user may select a downloaded file to upload; after doing so, clicking either submit button uploads it.
- Clear This button removes all rule blocks.

3 Rules

To clarify our discussion, we focus on the `Tense` block in the Latin theory, in particular, this rule:

```
future–bi: «cat» {conj1,conj2} «morph» {Future} «seg» 0 –> ɪb / _>
```

Our discussion will define these terms:

```
block: block name, rules
rule: name (optional), components
component: condition or change
condition: tier, constraints
constraint: bracketed list of values or bracketed pattern
change: tier, target, replacement, environment (optional), flags (optional)
environment: previous context, subsequent context
target and context: elements
element: IPA+diacritics, feature list, shorthand, null, ...
```

3.1 Conditions

The `future-bi` rule above has two conditions. The first is in the `«cat»` tier with the single constraint `{conj1,conj2}`. This constraint is only met if that tier contains either `conj1` or `conj2`. The second condition is in the `«morph»` tier; its single constraint is only met if that tier contains the MP `Future`.

Within a constraint, a value may be negated, such as `{-conj4}`. A negated value matches only if that given value is not in the current form.

Beside a bracketed list of values, a constraint may be a bracketed pattern. For instance, the constraint `{/1/}` in tier `«morph»` matches `1sg` and `1pl`, but not `3sg`.

If all the constraints in a rule are met by the current form, the rule is **available**. The Pāṇini **precedence** of the set of constraints in a rule is $\sum_j 1/size_j$, where the size of a constraint is the number of values it contains. If a rule has no constraints, its Pāṇini precedence is a minimal positive number. The `future-bi` rule has Pāṇini precedence 1.5. As it applies a block to a form, the Morphoton selects that available rule with the highest Pāṇini precedence. If there is a tie, the Morphoton complains about the ambiguity.

Occasionally one needs to artificially raise the Pāṇini precedence of a constraint to avoid ambiguity. The easiest way to do that is to duplicate one or more of the constraints.

3.2 Changes

Changes are applied to the current form, typically to the «seg» tier. A rule may have several changes. They are applied sequentially to the form. We say that a change is **effective** if its target and environment match the form and if its replacement modifies the form. If any of the changes in a rule fails to be effective, all the changes are blocked and the form remains as it was.

A change component has these constituents.

- **tier**, such «seg». If a rule has only one component, then «seg» is assumed.
- **name**, such future-bi. If a name is missing, the Morphoton assigns it the name of its block. Names are only important for documentation, so they need not be unique.
- **target**, such as 0
- **replacement** after an arrow (→ or ->), such as [-tense]
- optional **environment** introduced by '/', such as _>. The environment must have a single underscore '_'. The (optional) material before the underscore constitutes the **previous context**, and the (optional) material after it constitutes the **subsequent context**.
- optional **flags** introduced by '//', such as exclusive. We discuss flags in Section 4.6.1.

The target and the context constituents are composed of a sequence of **elements**, the concatenation of which is matched against the current form. A change matches a form if its target, surrounded if specified by the previous and subsequent contexts, matches the form. It can match in more than one place in the form. If a rule matches a form, the Morphoton replaces all sequences in the form that match the target with the replacement.

3.2.1 Elements

An element may be

- an IPA letter, with optional diacritics, representing a segment, such as ʁ or p͡f or á, Such an element matches the identical segment in the form.
- a feature list: a bracketed, comma-separated list of features with polarities, such as [+consonantal,-continuant]. The polarities are usually + or −; we discuss other polarities in

Section 4.6.2. A feature list matches any segment in the form that agrees on the given features, ignoring any feature that it does not explicitly list.

- a shorthand: `V` for "vowel", `C` for "consonant", `X` for "any defined segment", `S` for "white space", and `U` for "subscript". Subscripts arise in autosegmental phonology, discussed in Section 4.7.
- Null: `∅` (Unicode 2005; one may use the ASCII `0` instead.) The null element matches between segments in the form.
- a custom phoneme, typically a Greek letter, such as `μ` or a capital letter, such as `A`. We discuss custom phonemes in Section 4.2; they act as abbreviations for sets of features.
- a punctuation symbol, such as `=` to indicate an enclitic and `⇒` to indicate a proclitic. Appendix 6.3 lists the standard punctuation symbols.
- various components of Perl pattern-matching regular expressions (regexps), such as parentheses and repetition marks (`*`, `+`, and `?`).
- abbreviations for more complex Perl regexps: `^` (the start of the form), `$` (the end of the form), `#` (the start or end of a word in the form), `<` (the start of a word), `>` (the end of a word).
- **Ellipsis** extends Perl pattern matching. The form `⟨...⟩` matches any expression surrounded by the given brackets, even if there are enclosed bracketed expressions. The brackets may be any Unicode pair except for parentheses and `{ }`, which have other meanings. Some examples: `[ ]` and `⌊ ⌋`. Similarly, the form `⟨...abc⟩` matches any bracketed expression that contains, in order, `a`, `b`, and `c` (which must be IPA segments). For example, consider these rules

  ```
  hx: ⟨...hs⟩ → ∅
  ha: ⟨...ha⟩ → ∅
  hn: ⟨...hn⟩ → ∅
  ```

  with current form `this⟨⟨that⟩then⟩`. The `hs` rule makes no change. The `ha` rule results in the form `this⟨then⟩` . The `hn` rule results in `this`.
- **Indexing** extends Perl pattern matching. A pattern in the *target* may include a superscript (limited to `¹²³⁴`) following a segment-like part of the target to index it. A segment-like part is any of these:
  - A segment: an IPA symbol with optional diacritics
  - A bracketed list of features, such as `[+voice,−syllabic]`.
  - A parenthesized region, such as `(θɪs)`.
  - A shorthand: `V`, `C`, or `X`. (Shorthands are implicitly parenthesized.)

The index can then appear in the change to refer back to the part of the form that matches the indexed part. For example,

```
reduplicate: <(CV)¹C² → ²ə¹¹²
```

changes `fonim` to `nəfofonim`.

- **First occurrence** extends Perl pattern matching to match the first occurrence of a pattern in the target or subsequent context. There are two versions of the feature:
  - `ᶠ([+coronal]&[+trill])` matches the first `[+coronal]`; if there is none, the pattern does not match. If it finds an appropriate segment, it then succeeds only if that segment also matches `[+trill]`. The outer parentheses are required.
  - `ᶠ[+coronal]` matches the first `[+coronal]`; if there is none, the pattern does not match. The pattern may be surrounded by parentheses or brackets; Shorthands `V`, `C`, and `X` are implicitly parenthesized.

  For example,

  ```
  Rule1: x → g / _ᶠ([+coronal]&[+trill])a
  Rule2: x → g / _ᶠ[+coronal]a
  ```

  If the underlying representation includes `axaaalaa` and `axaaaraa`, then `Rule1` only applies to `axaaaraa`, because the first `+coronal` after `x` (`l` or `r` in the two underlying representations) must also be `+trill`, which is only true of `r`. However, `Rule2` applies to both underlying representations.

3.2.2 Replacements

The replacement may be

- `∅`; such a replacement simply deletes the matching target.
- a sequence of IPA letters with optional diacritics.
- a bracketed set of features, such as `[+voice,−tense]`. Such a replacement is only appropriate if the target matches a single segment. It modifies that segment by changing the polarities of the given features as shown, if possible. If no valid segment results, no change occurs. The Morphoton accepts several special polarities that only apply in replacements; we discuss them in Section 4.6.2.

4 Advanced

The discussion above covers most of the Morphoton's abilities. However, we have implemented other techniques for special purposes.

4.1 Comments and literate programming

Comments may appear in the **Preamble** and **Rule blocks** sections. A comment starts with the `%` symbol and continues to the end of the line.

Comments that start with `%!` have an additional purpose: **literate programming**. The Morphoton collects literate comments that appear the start of a block (in the lines immediately following the block name, including the `Morphology` and `Roots` blocks in the **Preamble** section and all rule blocks) into an output section called **Discussion**. It also collects literate comments following individual rules and show them as a list of items in the Discussion under their block name. To emphasise a word or phrase in a literate comment, surround it by asterisks `(*)`.

4.2 Phonology

The `Phonology` region in the **Preamble** section allows the theory to modify the default phonemic inventory. Each modification is in a single line, starting with a keyword.

- `Inventory`. Follow this keyword by a space-separated list of IPA symbols (and diacritics if needed) to defines a custom inventory of symbols that replaces the default phonemic inventory, which is based on the IPA.

  If a rule would produce a segment outside the inventory, the rule is ineffective; any changes (even unrelated to the failed change) are abandoned.

  The Morphoton has a region that is typically hidden called **IPA Chart** that lists all the IPA symbols. When you click on one or more symbols, the Morphoton displays a pane that shows the phonological features of the selected symbols. If the theory only uses forms employing IPA symbols without diacritics, it is unnecessary to specify an inventory. But if any segments need diacritics, then this line is required and must include all segments, including standard IPA symbols, along with any diacritics, such as `Inventory: a á ã b t x`. Also include in the inventory any unusual punctuation symbols that you use, such as `✥`.

The shorthand `<all>` introduces all IPA symbols without diacritics. So you can write `Inventory: <all> á ã`.

Another shorthand adds diacritics that are associated with phonological features to a set of IPA symbols. It has this form: `<a,i,u,ɪ><tone_high,long;tone_low>`. The available features include secondary articulations, such as `velar`. Spaces are not allowed in this shorthand. All the IPA symbols in the first delimited section (here `a,i,u,ɪ`) are entered into the inventory along with diacritics specified in the second section. That section has semicolon-delimited subsections (`tone_high,long` and `tone_low`). Each subsection names features; they can be either ordinary phonological features or secondary articulations, such as `velar`.

Each subsection is independently applied to the given IPA symbols. If it lists n features, the result is all 2^n combinations of those features. In the example above, the result is `a á aː áː i í iː íː u ú uː úː ɪ ɪ́ ɪː ɪ́ː à ì ù ì̀` .

- `Define`. To introduce **custom phonemes**, follow this keyword by a space-separated list of letters, typically capital Latin letters (but not `XVCSU`, which have special meaning) or Greek letters (but not `αβγδ`, which have special meaning). These letters are intended to be used as morphophonological segments in roots and intermediate, but not surface, forms. One can annotate each letter to establish phonological features.
  - No annotation. The letter is underspecified for all phonological features.
  - A bracketed list of phonological features with polarity, such as `A[+syllabic,-low]`. Omitted features are underspecified. Valid polarities are `+-0⊗`. The polarity `⊗` means explicitly underspecified. We discuss matching underspecified features in 4.6.2. If you click an IPA symbol in its chart, the pane that pops up has a section called `Features`. You can click that section to copy it to the clipboard, and you can then paste the result in the appropriate line in the preamble. In this way you can easily create custom morphemes that are similar to existing IPA symbols but have significant differences. However, the next method is often better.
  - Inheritance from an existing phoneme with modification, such as `A=a[+tense]`. The custom phoneme acquires all the features and polarities of the existing phoneme, with modifications as shown. The Latin theory uses this technique.

  Don't include custom phonemes in the inventory specification.

  The Morphoton automatically handles any related features. For example, `[+low]` implies `[-high]`, and `[+round]` implies `[+labial]`.
- `Redefine`. Follow this keyword by a single IPA symbol representing a segment, followed by a bracketed list of phonological features with polarity. This specification overrides the usual feature list for this segment, but the segment retains any features not mentioned in the list.

The Latin theory redefines the phoneme `a` so that it is `[-tense]`.

- `DisallowForm`. Follow this keyword by a space-separated set of patterns, much like the patterns in rule targets. Any rule that would result in a form whose `«seg»` tier matches one of these disallowed patterns fails to apply and is therefore not effective.
- `Delete`. Follow this keyword by a bracketed list of phonological features (without polarity) that the Morphoton should removes from its set of features. Such features may then not appear in any rule. Removing features can make multiple segments appear identical; for example, `Delete: voice` renders `t` and `d` indistinguishable.
- `Geometry`. Follow this keyword by a new name, `=`, then a space-separated list of features. For instance, `Geometry: StrTense = strident tense` introduces a new geometry called `StrTense` that refers to the two features, `strident` and `tense`. The Morphoton predefines several useful geometries:

name	feature set
cor	coronal anterior distributed strident
dor	dorsal high low front back tense
lab	labial labiodental round
place	cor dor lab
lar	constr_gl spread_gl voice

New geometries introduced by `Geometry:` may refer to these predefined geometries as well. We discuss the use of geometries in Section 4.6.2, where we also discuss the pseudo-polarity `∀`.

4.3 Morphology

A morphosyntactic property (MP), such as `Aspect`, can have one or more **binary** values, such as `+Caus` `+Repet`. If there are n binary values, then all 2^n combinations exist for this property, such as `+Cause,+Repet`, `+Cause,-Repet`, `-Cause,+Repet`, and `-Cause,-Repet`.

An MP name can be followed by `?` to indicate that its presence is optional. This feature is generally used for single-value MPs, and is therefore equivalent to using a binary value indicated by `+`.

It is valid to specify the absence of a value, whether binary or not, in the **Input values** box, by negating it. The Morphoton treats a negated value identically to a missing value.

4.4 Expected results

You can include expected results in a `Results` region in the **Preamble** section. Each line in that block specifies a lexeme (by its gloss), an MPS, and the expected result, separated by `/` marks, such as

```
warn / 1sg Present / ɔɛmno
```

If it analyzes that lexeme with that MPS, the Morphoton decorates the resulting surface form either with ✓ or ✘, depending on agreement with the expected result.

4.5 Conditions

Conditions are restrictions based on the current values of tiers. They are written as lists of tier names and bracketed expressions, such as

```
«cat» {conj1,conj2}{verb} «morph» {–neg}
```

The conditions in this example require that the `«cat»` tier have either `conj1` or `conj2` (or both), and it must have `verb`. In addition, the `«morph»` tier must not have `neg`.

Conditions may be placed on blocks (except for `(all)` blocks) and on rules. To place a condition on a block, place it after the block name on the same line. If a block's condition is not met, that block is skipped.

To place a condition on a rule, place it at the start of the rule, after the optional rule name.

Usually, a condition on a tier refers to the presence (like `{verb}`) or absence (like `{–neg}`) of values in that tier. For conditions on rules (but not on blocks), one can also write an MP value prefaced by `↓`, as in `{↓verb}`. This pattern matches just like `{verb}`, but if the rule is selected and is effective, then a side-effect of that rule is to remove that word from that tier. Similarly, one can write a value prefaced by `↑`, as in `{↑conj1}`. This pattern matches just like `{–conj1}`, but if the rule is selected and is effective, then a side-effect of that rule is to add that word from that tier.

If you want to remove a value if it is there, but not cause the condition to fail if it is not there, use `{↓value,–value}`. Likewise, to add a value, but not cause the condition to fail if it is already there, use `{↑value,value}`.

One can also write a pattern-based condition, such as `«morph» {/sg/}`. This condition matches, for example, the `«morph»` values `1sg`, `2sg`, and `3sg`.

In an `(all)` block, each rule is tested before it run to see if its conditions, if any, are satisfied. Because

rules can modify the contents of any tier, this test is delayed until just before each rule is attempted.

4.6 Changes

4.6.1 Flags

Change components in rules may include flags that modify the way the Morphoton applies the rules. By default, a rule applies every place in which it can within a form, even if those places overlap. Flags override this behavior. They start with `//` and are placed at the end of the change component, following the environment (if any).

- `once`: The change applies only once, in the leftmost possible location in the form, even if it could apply in other places as well. For example,

  ```
  intensify: V → [−tense] // once
  ```

  modifies `ibik` to `ɪbik`, modifying only the first `i`.

- `iterate`: The change applies repeatedly until it makes no further modifications to the form. For example,

  ```
  spreadTense: [+syllabic] → [+tense] / _C[+tense] // iterate
  ```

  modifies `ɔfɪfi` to `ofifi`; without iteration, the result would be `ɔfifi`.

- `exclusive`: The change applies in all places, left to right, but suppressing later applications if they apply in a region matched by an earlier application. For example,

  ```
  Simple onset: ∅ → ‹ / _C?V // exclusive
  ```

  changes `abracadabra` to `ab‹ra‹ca‹dab‹ra`; without exclusion, the result would be `‹ab‹r‹a‹c‹a‹d‹ab‹r‹a`.

4.6.2 Special polarities

Besides `+` and `−`, the Morphoton allows other polarities for phonological features in certain contexts.

- `±`: This polarity applies in the replacement of a change. It allows the Morphoton to select any segment in the inventory that modifies this feature, but only if necessary. For example,

  ```
  lenition: [−continuant,−nasal] → [+continuant,±delayed_release,±distributed]
  ```

replaces any non-nasal consonant by a continuant version of that consonant, ignoring the features `delayed_release` and `distributed` if necessary. For example, it replaces `t` by `θ`, even though this change also introduces `+delayed_release` and `+distributed`. It also replaces `c` by `ç`, which introduces `+delayed_release` but preserves `+distributed`.

- `⊕` and `⊖`: These polarities apply only in the replacement part of a change, where they indicate a preference to change the segment's polarity to `+` or `−`, respectively, but only if the result is in the inventory. For example,

  ```
  reduction: [+syllabic] → [−tense,⊖high]
  ```

  replaces `u` by `ɔ`, introducing `[−tense,−high]`. However, if we remove `ɔ` from the inventory, then it replaces `u` by `ʊ`, which is `[−tense,+high]`.

- `α`, `γ`, and `δ`: These polarities introduce **variables**. (The letter `β` is an IPA symbol, so it can't be used to name a variable.) Variables acquire their definition in the target of a change, that is, they become bound to `+` or `−` based on the feature that they match. A rule may reference that definition in its replacement and environment, and it may also negate that definition as `−α`, `−γ`, and `−δ`. For example,

  ```
  harmony: [αhigh] → [+back,−low,+round] / [+back,+round,αhigh]_
  ```

  replaces `i` by `u` after `u` (which is `+high`) and replaces `ɤ` by `o` after `ɔ` (which is `−high`).

- `⊗`: This polarity may only appear in the target or environment of a change. It matches a custom phoneme that is underspecified for the given feature. For example, consider a custom phoneme `T` that is like `t` except that it does not specify voice:

  ```
  Define: T[−syllabic,+consonantal,−sonorant,−continuant,−delayed_release,
    −approximant,−tap,−trill,−nasal,−spread_gl,−constr_gl,−labial,−round,
    −labiodental,+coronal,+anterior,−distributed,−strident,−lateral,
    −dorsal,0high,0low,0front,0back,0tense]
  ```

  The change

  ```
  matchVoice: [⊗voice] → [+voice] / [+voice]_
  ```

  replaces `T` by `d` after a voiced segment.

- `∀`: This polarity may only appear in the replacement or environment of a change. The associated feature must be a geometry feature. It acquires its definition in the environment, collecting the polarities of all the phonological features that underlie the geometry feature, as defined in 4.2. In the replacement, it refers to all those features with those polarities. For example,

  ```
  Nasal assimilation:  ŋ → [∀place] / _[∀place]
  ```

  replaces `ŋ` by `ɲ` before `ç` (whose `place` geometry asserts `[+coronal,−anterior,+distributed, −strident,+front,−back]`.

4.6.3 Conditional changes

Besides using a condition component in a rule, you can condition a change by placing an `IF–THEN–ELSE` sequence in the change after the tier name. For example,

```
pluralize: «seg» IF [+strident]> THEN ø → əs / _> ELSE ø → z / _>
```

replaces `kɪs` by `kisəs` and `kɪd` by `kɪdz`. The `IF`-part may be any sequence usable in the target. The `THEN` and `ELSE` parts may be any changes, including environments and options. The `ELSE` part (but not the `THEN` part) can even be another `IF–THEN–ELSE` sequence. The `ELSE` part is optional.

A similar condition is to place an `UNLESS–DO` sequence in the change after the tier name. For example,

```
pluralize: «seg» UNLESS [+strident]> DO ø → z / _>
```

replaces `kid` by `kɪdz` but does not modify `kɪs`.

4.6.4 Feature lists

Feature lists in the `«seg»` tier refer to phonological features. Similarly, you may use feature lists referring to the `«morph»` and `«cat»` tiers in changes, both the target and the replacement as well as the conditions in an `IF–THEN–ELSE` sequence.

In the `«morph»` tier, elements of such lists refer to MP values. In the `«cat»` tier, elements of such lists refer to category values. For example,

```
«morph» [+Present,–1sg] → [–Present] «cat» [+Conj3] → [–Conj3,+Conj1]
```

changes the `«morph»` tier by removing `Present`, but only if `1sg` is not asserted. In the «cat» tier, it replaces `Conj3` with `Conj1`. If either of these changes fails, then neither is effective. The only valid polarities on values in these tiers are `+` and `–`.

4.6.5 Implied features

When a change modifies a feature in the `«seg»` tier, that modification can imply other feature changes. The Morphoton automatically performs the implied modifications. Implications have several types.

1. Direct implication, for instance, `[+round]` implies `[+labial]`.
2. Licensing removal, for instance, `[–coronal]` implies `[0strident]`.
3. Anti-licensing, for instance, `[+sonorant]` implies `[0delayed_release]`.

4. Reverse licensing, for instance, `[+strident]` implies `[+coronal]`.

5. Exclusivity, for instance, `[+high]` implies `[−low]`.

The Morphoton does not automatically perform similar modifications in other tiers. For example, in the «morph» tier, if the `Tense` MP contains values `Past` and `Future`, a change that introduces `Future` does not automatically remove `Past`.

4.7 Autosegmental phonology

The Morphoton is able to deal with autosegmental phonology, first introduced by Bird and Ladd (1991). Besides that standard tiers, «seg» and «morph», one can introduce other tiers, in particular, «tone».

The roots may include multiple tiers. For example,

```
«seg» gaga da bala ma «tone» HL N LH HL
```

contains both a «seg» tier and a «tone» tier. If there are multiple tiers, each must contain the same number of (space-separated) words, because the Morphoton assume that they correspond. In the example above, for instance, the word `bala` in the «seg» tier corresponds to the tones `LH` in the «tone» tier.

Only a restricted set of symbols may appear in the «tone» tier: `THMLBthmlbN`. These refer to tones `top`, `high`, `medium`, `low`, and `bottom`, then their floating versions, then `unassigned`.

Autosegmental rules often have multiple change components. For example,

```
deduplicate: «seg» V¹ → ∅ / ¹_ «tone» H → N
```

simultaneously removes the second of a pair of identical vowels and reduces its tone to unassigned, but only if its tone was high.

In addition to ordinary change components, multiple-tier changes also respond to special autosegmental rules. Many of these rules take flags, signalled by `−−`. Most of these flags have default values, as shown here.

We illustrate these rules by considering the following theory:

```
Preamble
Inventory: <all> <a><tone_high,tone_low,tone_falling>

MPS
Person: demo % just to say something

Roots
test: «seg» gaga da bala ma «tone» HL N LH HL

Sandhi (all)
deduplicate: «tone» H → N / _S*H
Associate:
Spread:
Realize: --null=tone_low
```

The Morphoton shows the results of this theory as shown in Figure 1. It aligns words to express their correspondence.

demo

Root	test: gaga da bala ma
Associate	ga_1ga_2 da_0 ba_3la_0 ma_{45} H_1L_2 N_0 L_3N_0 H_4L_5
Spread	ga_1ga_2 da_0 ba_3la_3 ma_{45} H_1L_2 N_0 L_3N_0 H_4L_5
Realize	gágà dà bàlà mâ
surface form	gágà dà bàlà mâ

Figure 1: Autosegmental example

The `deduplicate` rule applies to the `«tone»` tier, replacing the `LH` (corresponding to `bala`) by `LN`.

The other rules are examples of the following.

- `Associate: --tiers=tone-seg --dir=L-R --features=[+syllabic] --endpoint=# --max=3`
  This rule indicates which tiers to associate, the direction of association within a chunk (either `L-R` or `R-L`), to which segments in tier_2 to associate, how to separate the chunks, and how many elements of tier_1 at most to associate with a segment in tier_2. Traditionally, links between tiers are shown with lines. Instead we show association by a common subscript in two tiers. So the first `a` in the `«seg»`

tier is associated with the first H in the «tone» tier. There are two tones, HL, to associate with the single vowel in the last word, ma; the result is shown with two subscripts: ma$_{45}$. The N tone is always associated with the subscript 0.

- Spread: --tier=seg --dir=L-R --endpoint=#
  This rule spreads associations in the given tier in the given direction. In our example, it spreads L$_3$ to associate with both vowels in bala, resulting in ba$_3$la$_3$ in the «seg» tier.

- Realize: --tiers=tone-seg --null=tone_low
  This rule uses tier$_1$ (typically «tone») to realize the associated values in tier$_2$ (typically «seg»). Unassociated segments acquire the value in the --null option, which has no default value and must be stated explicitly. In our example, the result is gágà dà bàlà mâ. The a in da is associated with the N tone, so it becomes tone_low. The final a is associated with two tones, H and L, so it aquires a falling tone.

The Morphoton recognizes three autosegmental rules not covered by the example in Figure 1. We demonstrate them in Figure 2, the Morphoton output for the following theory.

```
Preamble
Inventory: <all> <a,i><tone_high,tone_low,tone_rising>

MPS
Person: demo % just to say something

Roots
test: «seg» ti tiari «tone» L LHL

Sandhi (all)
Associate:
glide formation: «seg» [+syllabic,+high] → [−syllabic] / _.a
Reassociate: --max=2
Join:
Split:
Realize: --null=tone_low
```

- Reassociate: --tiers=tone-seg --dir=L-R --features=[+syllabic] --endpoint=# --max=1
  This rule moves associations that no longer apply to later (if L-R) or earlier (if R-L) segments. An association no longer applies if a rule has changed its segment so it no longer satisfies the features parameter. In our example, after the glide formation rule, link 2 is now associated with j, which

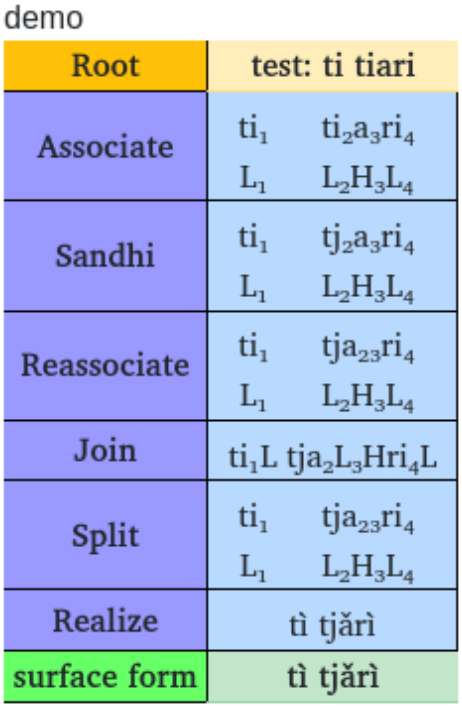

Figure 2: Autosegmental example

is not syllabic. `Reassociate` fixes that problem. It is necessary to override the default `max=1`, though, to allow two tones to link to the `a` in `tjari`, resulting in `tja₂₃ri₄`.

- `Join: --tiers=tone-seg`
  This rule places information from tier_1 into tier_2, resulting in just one tier, called `«tone-seg»` (for the default tiers). The purpose is so further rules can refer to the combined content. In our example, each subscript in the `«seg»` tier is followed by the associated tone in the `«tone»` tier.

- `Split:`
  This rule undoes the effect of the `Join` command, creating the original two tiers, so far as possible. It takes no options. In our case, no rules intervene between `Join` and `Split`, and the Morphoton is able to create the previous two-tier value.

5 User-interface features

The Morphoton has many features that assist the user. We discuss submitting data, expected results, and lesser-used regions of the interface.

5.1 Submitting data

After filling in the input regions, the user may click either Submit or Refresh-submit to submit the data to the server for evaluation. If the Morphoton has already computed results, it places a resubmit button before the results. The Submit and resubmit buttons employ AJAX, a web technique that avoids redrawing the entire page. The result simply appears in the results area. The browser keeps track of changes you have made in the input sections; you can usually undo them by using <ctrl-Z>. The Refresh-submit button performs a standard web submittal and builds a new page; you can use the browser's "back" arrow to return to the previous page.

For pedagogical purposes, some theories in the **Library** might be password-protected. The passwords are kept on the server.

5.2 Pronunciation and spectrograms

If you click on a surface form in the result table, the Morphoton sends the content to a web facility* that attempts to pronounce that form. The result is not necessarily high quality. (The web facility ignores stress and is imperfect in other ways.) In addition to a media-control interface, which allows you to replay the sound, the Morphoton also presents a new button, spectrogram, which you can click to send the resulting sound file to a Praat routine to generate a spectrogram of the sound. The result can be useful in demonstrating how spectrograms look.

5.3 Lesser-used regions

In order to keep the user interface simple, the Morphoton initially hides several regions. It presents a show/hide button that the user can click to expand or contract these regions. We discuss each of those regions in turn.

*`https://iawll6of90.execute-api.us-east-1.amazonaws.com/production`

5.3.1 Sample rules

We have found many rules to be helpful in building morphological theories. Some of these are visible if you expand **Sample rules**. The rules are categorized by general type, such as **Syllabification** and **Footing**. Each rule is presented as a button, such as `Labialization: C → [+round]`. Clicking on any such rule adds it to the end of the current block of rules.

5.3.2 Library

We have built morphological theories for some aspects of many languages, often based on textbook examples. These theories are available if you expand **Library**; they are arranged in a table. You can enter search terms like `clitic` or `Romance` in the Search box to reduce the number of visible theories. Each one names its language with a button, such as Latin; clicking on that button submits the theory to the Morphoton, which responds with a new page of results. The Metadata and comments in the **Rules** section describe the theory.

5.3.3 IPA chart

The IPA chart is a valuable tool to allow you to see the features underlying every IPA symbol, to compare the features of several symbols, to select symbols based on features, and to choose restricted inventories.

The chart is divided into Vowels, Consonants, Other symbols (like `ʍ`), Affricates (like `ʧ`), and Doubly-articulated stops (like `k͡p`). The layout of the vowels and consonants is meant to reflect standard organization, separating characteristics such as tongue positions (such as `front` for vowels and `dental` for consonants).

When you click any entry in the chart, it is selected and acquires a colored background, and its features are shown in a blue pop-out on the right side of the page. Click the entry again to unselect it. You can click the `Features` region of that pop-out to copy the list of features with polarity; you can then paste the list into a `Define` rule in the preamble.

If you click multiple entries in the IPA chart, the pop-out shows the features that distinguish those entries. For instance, if you click `t` and `d`, you see that `t` is `+voice`, whereas `d` is `−voice`. The list of features they share is also shown (and can be copied by a click).

If you have selected several IPA entries, you can click Make inventory at the top of the chart. IPA entries in the inventory are shown with blue-outlined cells, and a new or replacement `Inventory` line appears in

the **Preamble**. Once you have built an explicit inventory, selecting entries shows how those entries are distinctive within the inventory; that is, if they form a natural class. For instance, if the inventory contains only `p`, `b`, `t`, and `d`, then selecting `t` and `d` causes the blue pop-up to show not only that they differ with respect to `voice`, but that they have the distinction within the inventory of being `-labial`. Similarly, `p` and `t` are have the inventory-based distinction `-voice`. However, `p` and `d` do not form a natural class.

Below the IPA chart is a Select by features button, which leads to a pop-up menu listing every phonological feature. Each can be asserted to have polarity `+, -, 0`, or `any`. Initially, all are set to `any`. You can assert any combination of features and then click Select (at the end of the list, which scrolls up and down). For instance, if you select `+distributed -strident +sonorant`, you see that ɲ, ɻ, and ʎ are selected in the chart, and the associated blue pop-out frame shows their features.

5.3.4 Segments and features

The chart of segments and features is an alphabetical list containing a row for each segment and a column for each phonological feature. You can select segments here just as in the IPA chart; in fact, a selection in either is also displayed in the other and brings up the blue pop-out. The header showing the features also emboldens those features for which the selected segments differ.

6 Appendices

6.1 Features

The Morphoton uses a set of phonological features based on the standard list in Hayes (2009). These features include, for instance, `syllabic`, `sonorant`, and `labial`. Each segment is defined by whether its polarity for each feature is `+` (the feature is present), `-` (the feature is absent), or `0` (the feature is irrelevant, because it is licensed by an absent feature). For example, the IPA symbol `/n/` has these features: `[-syllabic, +consonantal, +sonorant, -continuant, 0delayed_release, -approximant, -tap, -trill, +nasal, +voice, -spread_gl, -constr_gl, -labial, -round, -labiodental, +coronal, +anterior, -distributed, -strident, -lateral, -dorsal, 0high, 0low, 0front, 0back, 0tense]`. The `dorsal` feature licenses `high` and other features, but `/n/` has `[-dorsal]`, so it has `[0high]`.

Many of the features are associated with diacritics in order to assert them in segments that ordinarily would lack them, such as making `/n̩/` syllabic.

In addition to Hayes's features, the Morphoton includes mutually exclusive tone features.

6.2 Diacritics

The diacritics are these:

`nasal`	◌̃ (Unicode 0303)
`syllabic`	◌̩ (Unicode 0329)
`constr_gl`	for vowels: ◌̰ (Unicode 0330); for consonants: ◌ʼ (Unicode 02bc)
`spread_gl`	for vowels: ◌̤(Unicode 0324); for consonants: ◌ʰ (Unicode 02b0)
`distributed`	◌̪ (Unicode 032a)
`front`	◌̟ (Unicode 031f)
`back`	◌̠ (Unicode 0320)
`round`	◌ʷ (Unicode 02b7)
`tone_top`	◌̋ (Unicode 030b)
`tone_high`	◌́ (Unicode 0301)
`tone_mid`	◌̄ (Unicode 0304)
`tone_low`	◌̀ (Unicode 0300)
`tone_bottom`	◌̏ (Unicode 030f)
`tone_rising`	◌̌ (Unicode 030c)
`tone_falling`	◌̂ (Unicode 0302)
`tone_highRising`	◌᷅ (Unicode 1dc4)
`tone_lowRising`	◌᷄ (Unicode 1dc5)
`tone_highFalling`	◌᷇ (Unicode 1dc7)
`tone_lowFalling`	◌᷆ (Unicode 1dc6)
`tone_peaking`	◌᷈ (Unicode 1dc8)
`tone_dipping`	◌᷉ (Unicode 1dc9)

6.3 Punctuation symbols

One may use any punctuation symbol in rules and underlying forms (except for `{}()[]`, which have special meanings). The following symbols have conventional meanings; you may use other symbols for punctuation, but you should introduce them in the `Inventory`.

Symbol	Unicode	Conventional meaning
=	u003d	enclitic
⇒	u21d2	proclitic
-	u2011	affix symbol (not a minus sign)
'	u20c8	stress symbol
⟨	u27e8	left infix boundary
⟩	u27e9	right infix boundary
≣	u2263	juncture
❮	u276c	left phrase boundary
❯	u276d	right phrase boundary
~	u0073	reduplicant boundary

7 Acknowledgements

The IPA charts are based on ones at `www.internationalphoneticalphabet.org/ipa-sounds/ipa-chart-with-sounds/`. The text-to-pronunciation tool is based on code at `ipa-reader.xyz/` by Katie Linero. The Morphoton builds spectrograms by invoking Praat (see `www.praat.org/`). The authors coded the Morphoton in Perl (Wall et al., 2000). The code is available under a Creative Commons CC-BY license.

References

Bird, S. and Ladd, D. R. (1991). Presenting autosegmental phonology1, *Journal of Linguistics* **27**(1): 193–210.

Hayes, B. (2009). *Introductory Phonology*, Wiley-Blackwell, Hoboken, NJ.

Wall, L., Christiansen, T. and Orwant, J. (2000). *Programming perl*, " O'Reilly Media, Inc.".