How to use the Phonomaton

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

September 10, 2026

1 Introduction

The Phonomaton is a web-accessible facility for entering phonological rules and underlying representations and automatically deriving the surface forms resulting from applying those rules to those representations. It is intended both as an aid in teaching phonology and to help researchers formalize phonological theories. It is heavily based on the segments and features discussed by Hayes Hayes (2009).

One can use the Phonomaton in several ways.

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

The Phonomaton 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 Phonomaton computes results of a phonological theory, use the top navigation bar to select **Library**, then click show/hide to expand that section. Click the Latin button. The browser submits this request, and the Phonomaton loads the stored theory into the inputs sections (**Metadata**, **Phonological rules**, **Morphological rules**, and **Underling representations**) and executes its instructions, showing the result in the **Latin Results** section.

The first column in the results chart lists the names of all the phonological rules. Each subsequent column shows a particular underlying form, such as `am-a:o:`, then the effect of each rule on that form, finally arriving at the surface form, such as `amō`. Any rule that does not modify the form presents an empty cell. We'll just call all these forms, whether underlying, intermediate, and surface, simply *forms*.

The results chart has a few features.

- If the table is too wide for the browser, there is a scrollbar 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 Phonomaton attempts to pronounce it.
- If you click Export LaTeX to clipboard, the Phonomaton converts the chart to a LaTeX table that you can insert into a LaTeX document.
- If the theory specifies expected results (in the **Underlying representations** section), they appear after the surface form, in green if they agree, in red if not.

There are five input sections.

- **Metadata**: optional extra information about the theory. The Metadata contains these fields:
  - Language, such as `Latin`
  - Date, such as `2023-02-28`
  - Language family, such as `Romance`
  - Code (Glottocode), such as `lati1261`
  - Sources, such as `Ørberg (2006)`
  - Implementation, such as `Daniel Kaufman`
  - Details, such as `length alternation, deletion`
  - Complexity, on a scale from 1 to 5, such `2`.

  The examples in the library follow these conventions.
- **Underlying representations**: forms to which rules are applied, with optional expected results. For instance, the first form is `am-a:`. The author of the theory expects that the resulting surface forms, after the Phonomaton applies the morphological rules and the phonological rules, should be `amō`,

`amās`, `amat`, `amāmus`, `amātis`, and `amant`. All forms generally use IPA representations for phonemes, along with IPA diacritics and various bracketing symbols. We call a phoneme plus its diacritics a *segment*. All diacritics are listed in Appendix 7.2.

- **Morphological rules**: modifications of underlying representations to derive related underlying representations. In the case of Latin, there are six rules, which produce six forms for the Phonomaton to address. By default, each underlying representation is also produced. The `ExcludeStem` keyword suppresses including the original underlying representation.
- **Phonological rules**: rules to sequentially apply to the derived underlying representations. For example, the `reduce` rule changes a long segment to a non-long segment if followed by at least one non-continuant consonant before the end of the word. This section should also contain comments (in lines starting with `%` to explain the method being used and the full citation of sources. Comments may start anywhere in a line and continue to the end of that line.
- **Buttons**: operations to Submit the theory for computation, to Clear the inputs, to Download the theory in an text format that can be saved, and to Browse local files for an text-format theory that you can then then upload on the subsequent Submit.

Some of these parts have a keyboard button, which gives you access to non-ASCII characters that you might want to use, particularly IPA symbols and diacritic marks.

3 Phonological and morphological rules

Except for preamble rules (see 4.1), the rules in the **Phonological rules** and **Morphological rules** sections indicate textual replacements. Each replacement rule has five components.

- the *name*, such `reduce`
- the *target*, such as `[+long]`
- the *replacement* after an arrow (→ or `->`), such as `[-long]`
- an optional *environment* introduced by '`/`', such as `_[+consonantal,-continuant]+>`. The *environment* must have a single underscore '`_`'. The material before the underscore constitutes the *previous context*, and the material after it constitutes the *subsequent context*.
- if desired, a set of *options* introduced by '`//`', such as `exclusive`. We discuss options in Section 4.2.

The *target* and the *context* parts are composed of a sequence of elements, the concatenation of which is matched against the current form. A rule 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 Phonomaton replaces all sequences in the form that match the *target* with the *replacement*.

3.1 Elements of patterns

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 current 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 4.3. A feature list matches any segment in the current 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 5.
- **Null**: `∅` (Unicode 2005; one may use the ASCII 0 instead.) The null element matches between segments in the current form.
- any defined custom phoneme, typically a Greek letter, such as `μ`. We discuss custom phonemes in Section 4.1; they act as abbreviations for sets of features.
- a punctuation symbol in the form, such as `=` to indicate an enclitic and `⇒` to indicate a proclitic, and several symbols for boundaries. Appendix 7.3 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,

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

with underlying representation `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. The *target* and *environment* may include multiple instances of an index (limited to ¹²³⁴). The first appearance of the index (left-to-right, starting with the *previous context*, then the *target*, then the *subsequent context*) defines the content of that index; it must follow a segment-like item, which can be 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 `(ðɪs)`.
  - A shorthand: `V`, `C`, or `X`.

  Subsequent appearances of the index refer to the content of that index. In the *change*, an appearance of the index refers to its content. Here it may be preceded by a feature list to modify the content for this appearance. For example,

  ```
  reduplicate: <(CV)¹C² → ²ə¹¹[-anterior]²
  ```

  changes `fonim` to `nəfofoɳim`;

  ```
  devoice: C → [-voice] / V¹_¹
  ```

  changes `ada` to `ata`.

- **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 Elements of replacements

The replacement may be

- `ø`; such a replacement simply deletes the matching target.
- a sequence of segments in the inventory.
- bracketed sets of features, such as `[+voice,-tense][-anterior]`. The number of features must be either 1, in which case they apply to all the segments in the target, or the same as the number of segments in the target, in which case each one applies to the corresponding segment in the target. Applying a feature set to a segment modifies that segment by changing the polarities of the given features as shown, if possible. If no valid segment results, no change occurs. The Phonomaton accepts several special polarities that only apply in replacements; we discuss them in Section 4.3.
- A combination of inventory segments and bracketed feature sets. The feature sets are applied as mentioned above, and the segments are inserted in corresponding positions. For example,

  ```
  double consonant: CC -> [+voice][-voice]o
  one change for both: CC -> [-anterior]
  and back: [-anterior]o -> [+anterior]
  ```

  If the underlying form is `atdl`, the first rule changes it to `adtol`; the second rule then produces `aḍṭol`, and the third rule results in `aḍtol` (no segment matches `o` with `[+anterior]`).

4 Advanced features of rules

4.1 Preamble

The Phonomaton accepts special "preamble" rules that limit the phonemic inventory, define new phonemes, disallow some forms, and delete some features. These rules start with keywords; unlike

replacement rules, they are not named.

- `Inventory:` This keyword is followed by a space-separated list of IPA symbols (and diacritics if needed) that defines a restricted inventory of symbols, replacing the full inventory that the Phonomaton starts with. If the forms only use IPA symbols without diacritics, this specification is not necessary. 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 `✥`.

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

  Another shorthand adds diacritics that are associated with 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 has features (they can be either ordinary 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:` This keyword introduces **custom phonemes**. It is followed 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 underlying and intermediate, but not surface, forms. Each letter may be associated with a bracketed list of features with polarity, such as `A[+syllabic,−low]`. If a feature is omitted, it is treated as underspecified. Alternatively, a new letter may inherit all the features of an existing phoneme, with modifications, such as `A=a[+tense,0tone_low]`. The polarities may be any of `0+−⊗`.

  The Phonomaton automatically handles any related features. For example, `[+low]` implies `[−high]`, and `[+round]` implies `[+labial]`.

  The features may include geometry clusters (see 4.1), but only with polarity `⊗`.

  Don't include custom phonemes in the inventory specification.

- `Redefine:` This keyword is followed by a single IPA symbol representing a segment, followed by a bracketed list of 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 features may include geometry clusters (see 4.1), but only with polarity `⊗`.

- `Disallow:` This keyword is followed by a space-separated set of patterns, much like the patterns in *targets* of rules. Any rule that would result in a form that matches one of these disallowed patterns fails to apply.

- `Delete:` This keyword is followed by a bracketed list of features (without polarity) that the Phonomaton then removes from its set of features. Such features may not appear in any rule. Removing features can make multiple segments appear identical; for example, `Delete: voice` makes `t` and `d` indistinguishable.

- `Feature:` This keyword is followed by a name of a new feature followed by specifications for all segments. For example, `Feature: ATR +<i,e,o,u> −<ɪ,ɛ,ɔ,ʊ,a> 0<default>` introduces a new feature called `ATR` which provides positive polarity for `i,e,o,u`, negative polarity for `ɪ,ɛ,ɔ,ʊ,a`, and 0 polarity for all other segments.

- `Geometry:` A *geometry cluster* is a set of features, typically related in some way to each other. The Phonomaton predefines several useful geometry clusters:

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

  You can introduce a new geometry cluster with the keyword `Geometry:` followed by a new name, `=`, then a space-separated list of features. For instance, `Geometry: StrTense = strident tense` introduces a new geometry cluster called `StrTense` that refers to the two features, `strident` and `tense`. New geometry clusters introduced by `Geometry:` may refer to these predefined geometry clusters as well. Geometry clusters can often be treated just the same as individual features. One such use is in in Section 4.3 when we discuss the pseudo-polarity `∀`.

4.2 Options

Phonological and morphological rules may include options that modify the way the Phonomaton applies the rules. By default, a rule applies every place in which it can within a form, even if those places overlap.

Options override this behavior. They start with `//` and are placed at the end of the rule, following the environment (if any).

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

  `intensify: V → [−tense] // once`

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

- `iterate`: The rule applies repeatedly until it makes no further changes. For example,

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

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

- `exclusive`: The rule 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`.

4.3 Special polarities

Besides `+` and `−`, the Phonomaton allows other polarities in certain contexts.

- `±`: This polarity applies in the *change*. It allows the Phonomaton 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 appear only in the *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*. Variables acquire their definition in the *target*, that is, they become bound to + or − based on the feature that they match. A rule may reference that definition in the *change* and the *environment*, and it may also negate that definition as −α, −γ, and −δ. The letter β is an IPA symbol, so it can't be used to name a variable. For example,

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

  changes i to u after u (which is +high) and ɤ to o after ɔ (which is −high).

- ⊗: This polarity may appear in the *target*, *environment*, and *change*. In the first two, it matches a custom phoneme that does not define the given feature. For example,

  ```
  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]
  % T is like t and d but does not specify voice
  matchVoice: [⊗voice] → [+voice] / [+voice]_
  ```

  changes T (which is underspecified for voice) to d after a voiced segment.

  In the *change*, the ⊗ polarity renders the given feature undefined.

  In the *change*, the ⊗ polarity along with a geometry cluster causes all the features associated with that geometry cluster to be undefined.

- ∀: This polarity may only appear in the *change* or *environment*. The associated feature must be a geometry cluster (See 4.1). In the environment, it acquires a definition, collecting the polarities of all the features in that cluster. In the change, it refers to all those features with those polarities. For example,

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

  changes ŋ to ɲ before ç (which asserts [+coronal, −anterior, +distributed, −strident, +front, −back], all of which are features of the place geometry cluster.

  When the ∀ polarity appears in the *environment* and the *change*, the polarity ⊗ may appear with that geometry cluster in the *target* as well. In that case, only the features in that cluster that are unspecified in the target are set to the polarities found in the *environment*.

4.4 Disjunctive rules

A rule can be conditioned on whether the current form matches a given pattern. For example,

```
pluralize: IF [+strident]> THEN ∅ → əs / _> ELSE ∅ → z / _>
```

changes `kɪs` to `kisəs` and `kɪd` to `kɪdz`. The condition may be any pattern. The `THEN` and `ELSE` parts may be any rules, including environments and options. The `ELSE` part is special: it may be omitted (in which case the condition failing leads to the rule not triggering) and it may be, recursively, a disjunctive rule. The `THEN` part must not be disjunctive.

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

```
pluralize: UNLESS [+strident]> DO ∅ → z / _>
```

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

Disjunctive rules are intended for morphological rules, building morphological forms that depend on the phonology of stems. However, they are also valid for phonological rules.

4.5 Comments and literate programming

Comments may appear in the **Phonological rules**, **Morphological rules**, and **Underlying representations** 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 Phonomaton collects literate comments in each section and displays them in a Discussion section at the top of the results page. To emphasise a word or phrase in a literate comment, surround it by asterisks `(*)`.

5 Autosegmental phonology

The Phonomaton is able to deal with autosegmental phonology, first introduced by Bird and Ladd (1991). Forms are subdivided into **tiers** named `«seg»` , `«tone»`, and `«morph»`. These tiers are meant to represent IPA segments, tones, and morphosyntactic feature sets. Forms that do not explicitly name tiers are assumed to be in the `«seg»` tier.

The underlying representation 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 they are assumed to 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`.

Rules, both in the **Phonological rules** and **Morphological rules** sections, may refer to a single tier or to multiple tiers. If the current form has multiple tiers, the rule must name the tier(s) to which it applies. A rule that refers to multiple tiers only takes effect on forms the trigger changes in all those tiers. 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 rules, multiple-tier forms also respond to special autosegmental rules. Many of these rules take options, signalled by `--`. Most of these options have default values, as shown here.

We illustrate these rules by considering the following theory:

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

[rules]
deduplicate: «tone» H → N / _S*H
Associate:
Spread:
Realize: --null=tone_low
```

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

underlying representation	gaga HL	da N	bala LH	ma HL
deduplicate	gaga HL	da N	bala LN	ma HL
Associate	ga_1ga_2 H_1L_2	da_0 N_0	ba_3la_0 L_3N_0	ma_{45} H_4L_5
Spread	ga_1ga_2 H_1L_2	da_0 N_0	ba_3la_3 L_3N_0	ma_{45} H_4L_5
Realize	gágà da_0 bàlà mâ			
surface form	gágà da_0 bàlà mâ			

Figure 1: Autosegmental example

The `deduplicate` rule applies only to the `«tone»` tier, changing the `LH` (corresponding to `bala`) to `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. The result of each association is shown by a subscript linking an element of tier_1 with an element of tier_2. The special tone value `N` always associates with subscript 0. In our example, each tone is associated with a single vowel in the `«seg»` tier, as shown by corresponding subscripts in both tiers. There are two tones, `HL`, to associate with the single vowel in the last word, `ma`; the result is two subscripts: $\texttt{ma}_{4\,5}$. 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 $\texttt{L}_3$ to associate with both vowels in `bala`, resulting in $\texttt{ba}_3\texttt{la}_3$ 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.

There are three special autosegmental rules not covered by the example in Figure 1. We demonstrate them in Figure 2, which refers to the following theory.

```
[underlying]
«seg» ti tiari «tone» L LHL

[rules]
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

underlying representation	ti L	tiari LHL
Associate	ti_1 L_1	$ti_2a_3ri_4$ $L_2H_3L_4$
glide formation	ti_1 L_1	$tj_2a_3ri_4$ $L_2H_3L_4$
Reassociate	ti_1 L_1	$tja_{23}ri_4$ $L_2H_3L_4$
Join	«tone-seg» ti_1L $tja_2L_3Hri_4L$	
Split	ti_1 L_1	$tja_{23}ri_4$ $L_2H_3L_4$
Realize	tì tjǎrì	
surface form	tì tjǎrì	

Figure 2: Autosegmental example

parameter. In our example, after the `glide formation` rule, link 2 is now associated with `j`, which is not syllabic. `Reassociate` fixes that problem. It is necessary to override the `max` parameter, though, to allow two tones to link to the `a` in `tjari`, resulting in `tja`$_{2\,3}$`ri`$_4$.

- `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, as far as possible. It takes no options. In our case, no rules intervene between `Join:` and `Split:`, and the Phonomaton is able to create the previous two-tier value.

6 User-interface features

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

6.1 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 Phonomaton 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>. Another option is to type <ctrl-enter> anywhere on the page, which acts exactly like a Submit button.

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. This action is necessary if you want to see the result of changing the inventory as you look at the IPA chart.

6.2 Surface forms

The following features are currently unavailable; they depend on an external web site that converts IPA into sound files. If you click on a surface form in the result table, the Phonomaton sends the content to a web facility that attempts to pronounce that form. The result is not necessarily very 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 Phonomaton 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.

6.3 Expected results

If the **Underlying representations** does not include expected results (such as `am-a: // amō / amās / amat / amāmus / amātis / amant` for Latin) and the Phonomaton has produced a table of results, it also presents a set expected button at the head of the **Underlying representations** region. Clicking that button fills in expected results from the table.

6.4 Lesser-used regions

In order to keep the user interface simple, the Phonomaton initially hides several regions of its interface. 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.

6.5 Sample rules

We have found many rules to be helpful in building phonological theories. Some of these are visible if you expand **Sample rules**. The rules are categorized by general type, such as **Phonological** and **Syllabification**. 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 set of phonological or morphological rules (whichever is appropriate).

6.6 Library

We have built phonological theories for some aspects of many languages, mostly based on textbook examples. These are available if you expand **Library**. These theories are arranged in a table. You can enter search terms like `syllabification` 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 Phonomaton, which responds with a new page of results. The Metadata and comments in the **Phonological rules** section describe the theory.

6.7 ACD

The Phonomaton can be used to explore historical linguistics. The massive compendium of Austronesian languages compiled by Robert Blust, Stephen Trussel, and Alexander D. Smith (2023) provides a fertile ground for such exploration. The **ACD** section of the Phonomaton presents a table of about 1,000 entries. Each button refers to an unfinished theory relating a modern language (such as `Abaknon`) to one of several proto languages (such as `PMP`). Unfinished theories only contain underlying representations (from the proto language) along with associated expected surface forms (from the modern language), along with a comment glossing the term into English. You can click the proto-language button, PMP for instance, to submit the unfinished theory for this combination to the Phonomaton. For `PMP–Abaknon`, we find, among other underlying forms,

```
qahəlu // allo % pestle
```

This form indicates that the PMP form `*qahəlu` ‘pestle’ is expressed as `allo` in Abaknon. The **Rules** sections are empty. Interested scholars can develop rules that generate the modern forms from the proto forms.

6.8 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 lavender 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 `Inventory:` line appears at the start of the **Phonological rules**. 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 lavender 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.

The Select inventory button inspects the current `Inventory` line in the **Phonological rules** section and uses its information to select segments in the IPA chart. It surrounds each selected segment with a blue box. However, if the inventory includes a segment modified by a diacritic, it surrounds the selected segment with a red box. If you hover the cursor over a red box, a tooltip shows you the variety (or varieties) of that segment with diacritics.

Below the IPA chart is a Select by features button, which leads to a pop-up menu listing every 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, ɹ, and ʌ are selected in the chart, and the associated blue pop-out frame shows their features.

6.9 Segments and features

The chart of segments and features is an alphabetical list containing a row for each segment and a column for each 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.

6.10 Pop-out results

If the Phonomaton shows results of a computation, it also presents a pop out button. Pressing this button brings up a new window that only contains the results of the computation. Whenever this new window is present, any use of a submit button or <ctrl-enter> (using AJAX) updates both the results section and the information in the new window.

6.11 Instructions

The instructions are a concise summary of the features of the Phonomaton. They show the Unicode equivalent of non-ASCII characters, including diacritics and their associated features.

7 Appendices

7.1 Features

The Phonomaton uses a set of *features* based on a standard list (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 Phonomaton includes mutually exclusive tone features.

7.2 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)

7.3 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

References

Hayes, B. (2009). *Introductory Phonology*, Wiley-Blackwell, Malden, MA.