phonokit

A toolkit to create phonological representations

MARCH 2026

Guilherme D. Garcia | gdgarcia.ca

What is it?

- A Typst package for phonology (Garcia, 2026)
- **Idea**: generate phonological representations (IPA, prosody, SPE, OT, etc.)
- Current version: `0.4.1`

phonokit

Typst...? Typst is a new language to typeset documents. It's modern, light, fast and intuitive. Visit typst.app to use their online editor (also check out their excellent tutorials)

Phonetic transcription with `#ipa()`

☞ Charis SIL is the default font, but you can alter it.

```
#ipa("[tR \\~ a Ns.kRi.'s \\~ a \\~ w]")
```

[tʀãŋs.kʀi.ˈsãw̃]

```
#ipa("['lIt \\v l \\s 'b2R \\schwar ,flaI]")
```

[ˈlɪtl̩ ˈbʌɾɚˌflaɪ]

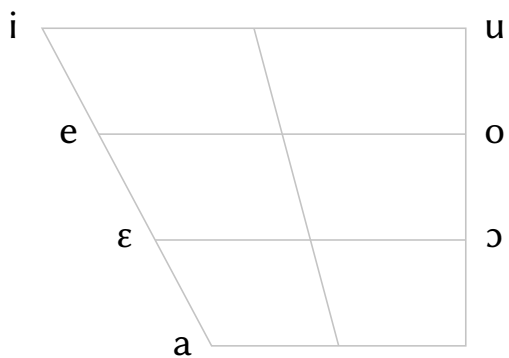
Intuitive shortcuts. Based on LaTeX's `tipa` package + some *very* subjective optimization:

- [ɲ]: `\textltailn` (LaTeX) → `\\nh` (both work in **phonokit**)
- [ɟ]: `\textbardotlessj` → `\\barredj`

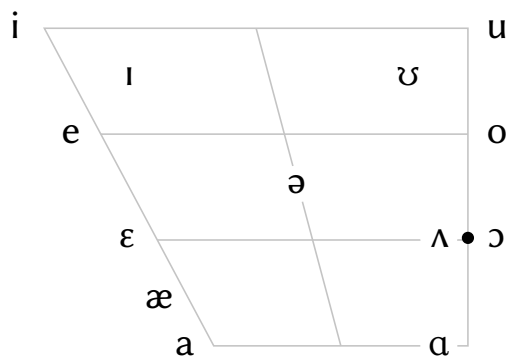
Phonemic inventories with `#vowels()` and `#consonants()`

- Vowel trapezoids (input = string): **pre-defined** inventories

```
#vowels("portuguese")
```



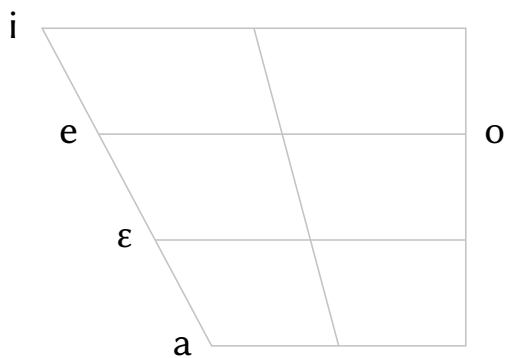

```
#vowels("english")
```



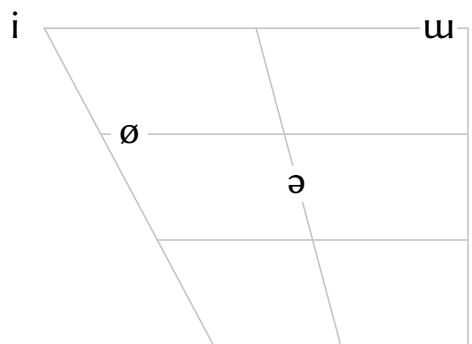
Phonemic inventories with `#vowels()` and `#consonants()`

- Vowel trapezoids (input = string): **custom** inventories

```
#vowels("aeoiE")
```

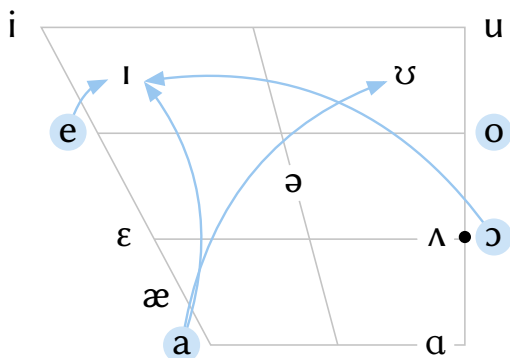



```
#vowels("\\o iW@")
```



Phonemic inventories with `#vowels()` and `#consonants()`

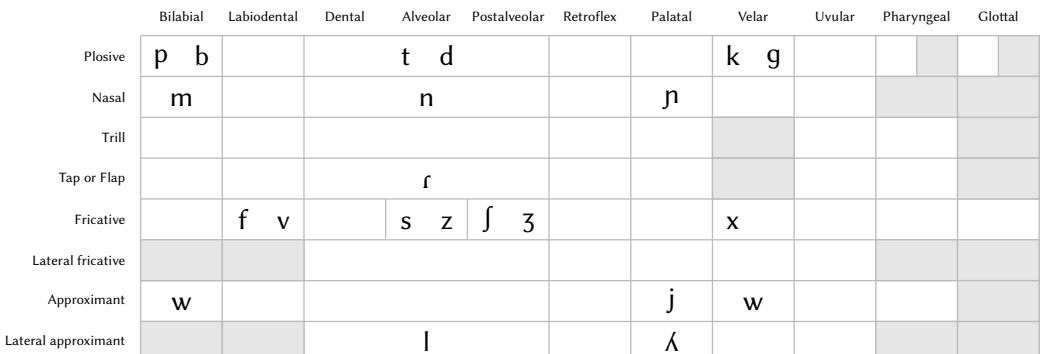
```
#vowels(
    "english",
    arrows: (
      ("a", "U"),
      ("a", "I"),
      ("e", "I"),
      ("O", "I"),
      ("o", "U"),
    ),
    arrow-color: blue.lighten(60%),
    curved: true,
    highlight: ("a", "e", "o", "O"),
    highlight-color: blue.lighten(80%),
  )
```



Phonemic inventories with `#vowels()` and `#consonants()`

- Consonant table (input = string): **pre-defined** languages or **custom** inventory

```
#consonants("portuguese", scale: 0.6)
```



Phonemic inventories with `#vowels()` and `#consonants()`

- Consonant table (input = string): **pre-defined** languages or **custom** inventory

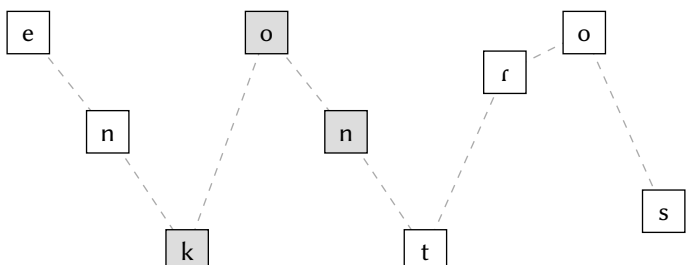
```
#consonants("french", scale: 0.6)
```

	Bilabial	Labiodental	Dental	Alveolar	Postalveolar	Retroflex	Palatal	Velar	Uvular	Pharyngeal	Glottal
Plosive	p b			t d				k g			
Nasal	m			n			ɲ				
Trill				r							
Tap or Flap											
Fricative		f v		s z	ʃ ʒ						
Lateral fricative											
Approximant	w						j	w			
Lateral approximant				l							

Visualizing sonority profiles with `#sonority()`

- Visual representation of the sonority principle (Parker, 2011)

```
#sonority("en.kon.tRos", scale: 0.7)
```

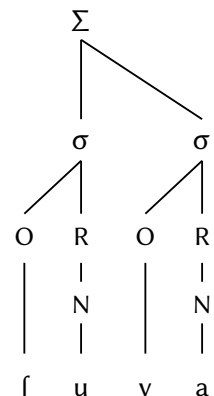


Prosodic representation with `#syllable()` and `#foot()`

- Syllable and metrical foot: intuitive functions to generate precise outputs

```
#syllable("maR")
```

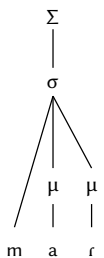

```
#foot("'Su.va")
```



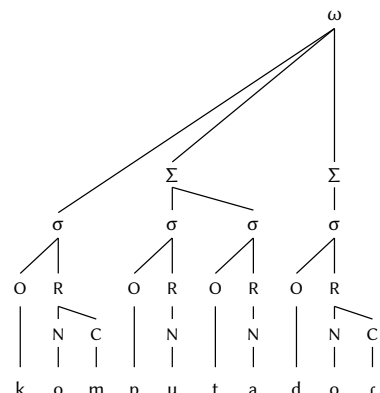
Prosodic representation with `#foot-mora()` and `#word()`

- Moraic representation and prosodic words

```
#foot-mora("maR", coda: true, scale: 0.68)
```



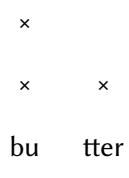

```
#word("kom.('pu.ta).('doR)", scale: 0.67)
```



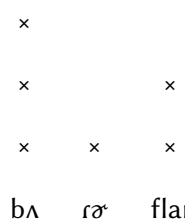
Prosodic representation: metrical grids with `#met-grid()`

- Input as string (left) or tuple with IPA support (right)

```
#met-grid("bu2.tter1")
```



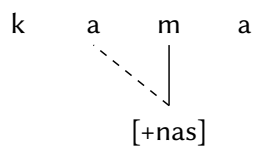

```
#met-grid(("b2", 3), ("R \\shwar", 1), ("flaI", 2))
```



Autosegmental phonology with `#autoseg()`

- Assimilation processes with intuitive and minimalist syntax

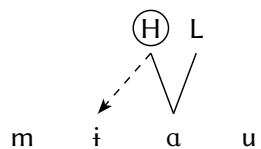
```
#autoseg(
    ("k", "a", "m", "a"),
    features: ("", "", "[+nas]", ""),
    links: ((2,1),),
    spacing: 1.0,
    arrow: false,
)
```



Autosegmental phonology with `#autoseg()`

- `#autoseg()` can easily be adapted to tonal processes with a wide range of arguments

```
#autoseg(
    ("m", "1", "A", "u"),
    features: ("", "", ("H", "L"), ""),
    tone: true,
    links: (((2,0),1),),
    highlight: ((2,0),),
    spacing: 1.0,
    arrow: true,
)
```



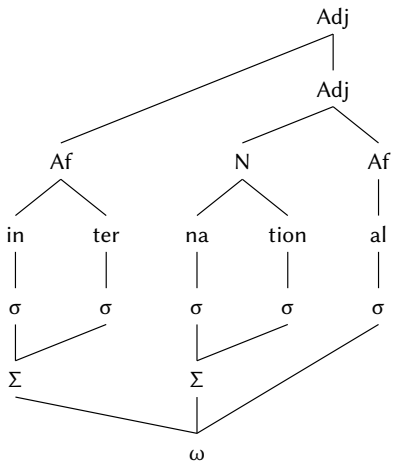
Autosegmental phonology with `#autoseg()`

```
#autoseg(
      ("e", "b", "e"),
      features: ("L", "", "H"),
      spacing: 0.5,
      tone: true,
      gloss: [],
    )
    #a-r // arrow
    #autoseg(
      ("e", "b", "e"),
      features: ("L", "", "H"),
      links: ((0, 2),),
      spacing: 0.5,
      tone: true,
      gloss: [èbě _pumpkin_],
    )
```

èbě *pumpkin*

Adapted from Zsiga (2024)

Multi-tier representations with `#multi-tier()`

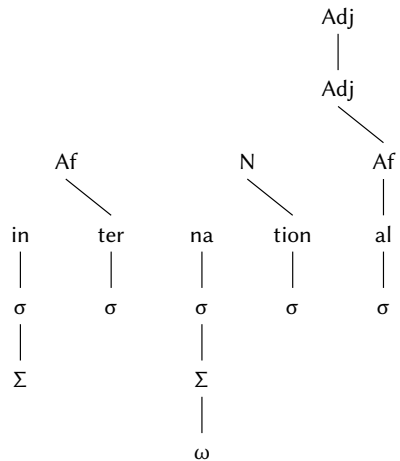


- Function `#multi-tier()` : **very flexible**
- Wide range of arguments based on a grid architecture
- Helper: temporary grid with coordinates
- Figure adapted from Booij (2012)

☞ Let's unpack this figure and its code

Multi-tier representations with `#multi-tier()`

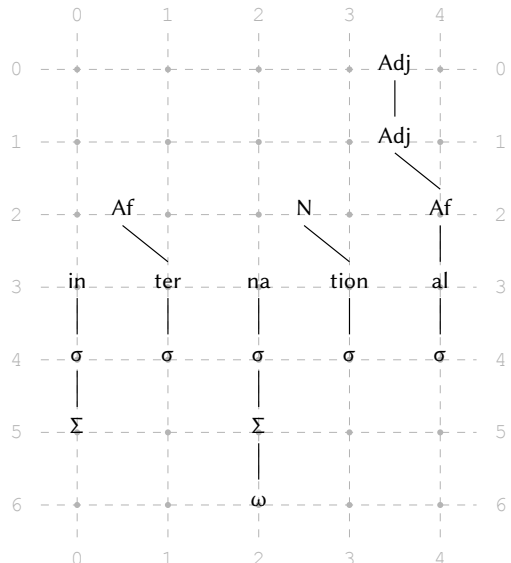
```
#multi-tier(
      show-grid: false,
      levels: (
        ("", "", "", "", ("Adj", 3.5)),
        ("", "", "", "", ("Adj", 3.5)),
        ("", ("Af", 0.5), "", ("N", 2.5), "Af"),
        ("in", "ter", "na", "tion", "al"),
        ("sigma", "sigma", "sigma", "sigma", "sigma"),
        ("Sigma", "", "Sigma", "", ""),
        ("", "", "omega", "", ""),
      ),
      scale: 0.8,
    )
```



- Any element projects **one** line/link by default (this can be deleted later with `delinks`)

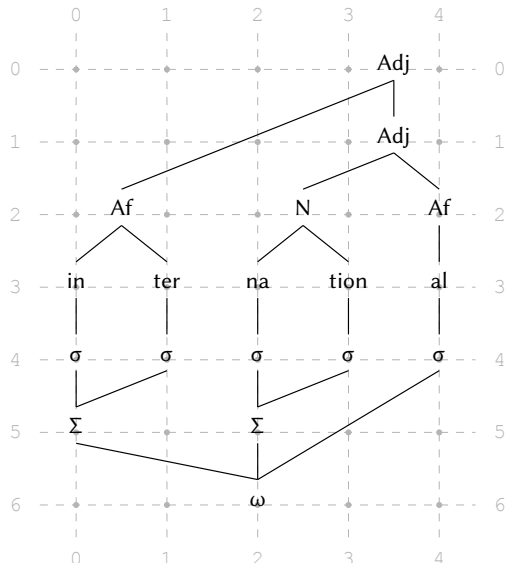
Multi-tier representations with `#multi-tier()`

```
#multi-tier(
      show-grid: true, // ← HELPER GRID
      levels: (
        ("", "", "", "", ("Adj", 3.5)),
        ("", "", "", "", ("Adj", 3.5)),
        ("", ("Af", 0.5), "", ("N", 2.5), "Af"),
        ("in", "ter", "na", "tion", "al"),
        ("sigma", "sigma", "sigma", "sigma", "sigma"),
        ("Sigma", "", "Sigma", "", ""),
        ("", "", "omega", "", ""),
      ),
      scale: 0.8,
    )
```



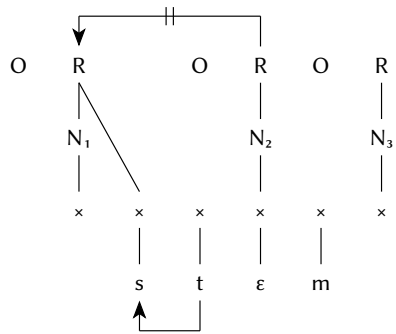
Multi-tier representations with `#multi-tier()`

```
#multi-tier(
      show-grid: true,
      levels: (
        ("", "", "", "", ("Adj", 3.5)),
        ("", "", "", "", ("Adj", 3.5)),
        ("", ("Af", 0.5), "", ("N", 2.5), "Af"),
        ("in", "ter", "na", "tion", "al"),
        ("sigma", "sigma", "sigma", "sigma", "sigma"),
        ("Sigma", "", "Sigma", "", ""),
        ("", "", "omega", "", ""),
      ),
      scale: 0.8,
      links: (
      ((0, 4), (2, 1)), // Adj → Af
      ((1, 4), (2, 3)), // Adj → N
      ((2, 1), (3, 0)), // Af → in
      ((2, 3), (3, 2)), // N → na
      ((5, 0), (4, 1)), // Ft → Syl
      ((5, 2), (4, 3)), // Ft → Syl
      ((6, 2), (5, 0)), // PWd → Ft
      ((6, 2), (4, 4)), // PWd → Ft
      ),
    )
```



Government Phonology with `#multi-tier()`

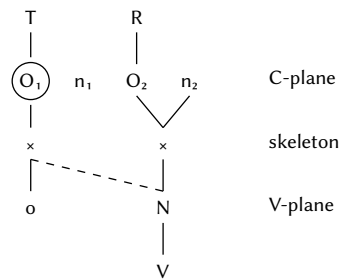
```
#multi-tier(
      levels: (
        ("O", "R", "", "O", "R", "O", "R"),
        ("", "N1", "", "", "N2", "", "N3"),
        ("", "x", "x", "x", "x", "x", "x"),
        ("", "", "s", "t", "E", "m", ""),
      ),
      links: (
        ((0, 1), (2, 2)),
      ),
      ipa: (3,),
      arrows: (
        ((3, 3), (3, 2)),
        ((0, 4), (0, 1)),
      ),
      arrow-delinks: (
        (1,)
      ),
      spacing: 1, scale: 0.8,
    )
```



Adapted from Goad (2012)

CV phonology with `#multi-tier()`

```
#multi-tier(
        levels: (
          ("T", "", "R", ""),
          ("O1", "n1", "O2", "n2"),
          ("x", "", ("x", 2.5), ""),
          ("o", "", ("N", 2.5), ""),
          ("", "", ("V", 2.5), ""),
        ),
        links: (((1, 3), (2, 2)),),
        dashed: (((2, 0), (3, 2)),),
        level-spacing: 1.2,
        highlight: ((1, 0),),
        spacing: 1,
        stroke-width: 0.7pt,
        tier-labels: (
          (1, "C-plane"),
          (2, "skeleton"),
          (3, "V-plane"),),
        scale: 1,
      )
```



Adapted from Carvalho (2017)

SPE with `#feat-matrix()` and `#feat()`

- Feature matrices for a given phoneme; matrices for rules + helper functions such as `#blank()`

```
#feat-matrix("\\ae")
```

/æ/

$$\begin{bmatrix} \text{+syllabic} \\ \text{−consonantal} \\ \text{+sonorant} \\ \text{+continuant} \\ \text{+voice} \\ \text{−high} \\ \text{+low} \\ \text{+front} \\ \text{−back} \\ \text{−round} \end{bmatrix}$$

```
#feat("+son", "-approx") #a-r #feat(alpha +
[#smallcaps("place")]) / #blank()\]#sub[#sigma] #feat("-
son", "-cont", "-del rel", alpha + [#smallcaps("place")])
```

$$\begin{bmatrix} \text{+son} \\ \text{−approx} \end{bmatrix} \rightarrow \left[\alpha\text{PLACE}\right] / \underline{\quad}\,]_{\sigma} \begin{bmatrix} \text{−son} \\ \text{−cont} \\ \text{−del rel} \\ \alpha\text{PLACE} \end{bmatrix}$$

OT with `#tableau()`

- Dynamic tableaux with auto shading (optional)

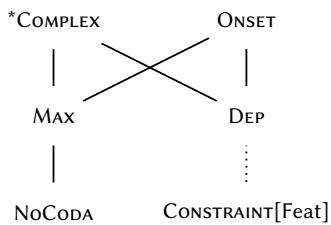
```
#tableau(
    input: "kraTa",
    candidates: ("kra.Ta", "ka.Ta",
"ka.ra.Ta"),
    constraints: ("Max", "Dep", "*Complex"),
    violations: (
    ("", "", "*"),
    ("*!", "", ""),
    ("", "*!", ""), ),
    winner: 0,
    dashed-lines: (1,),
    shade: true // ← auto shading after !
)
```

/kraθa/	Max	Dep	*Complex
☞ kra.θa			*
ka.θa	*!		
ka.ra.θa		*!	

Hasse diagrams with `#hasse()`

- Visualizing OT rankings with minimal syntax and with automatic small caps

```
#hasse(
    (
    ("*Complex", "Max", 0),
    ("*Complex", "Dep", 0),
    ("Onset", "Max", 0),
    ("Onset", "Dep", 0),
    ("Max", "NoCoda", 1),
    ("Dep", "Constraint[Feat]", 1, "dotted"),
    ),
    node-spacing: 3,
)
```



Harmonic Grammar with `#hg()`

- Weights and violation counts are used to automatically compute harmony scores (h_i)

```
#hg(
    input: "kraTa",
    candidates: ("[kra.Ta]", "[ka.Ta]",
"[ka.ra.Ta]"),
    constraints: ("Max", "Dep", "*Complex"),
    weights: (2.5, 1.8, 0.5),
    violations: (
      (0, 0, -1),
      (-1, 0, 0),
      (0, -1, 0),
    ),
    scale: 0.8,
    )
```

	w = 2.5	w = 1.8	w = 0.5	
/kraθa/	Max	Dep	*Complex	h_i
[kra.θa]	0	0	−1	−0.5
[ka.θa]	−1	0	0	−2.5
[ka.ra.θa]	0	−1	0	−1.8

Noisy Harmonic Grammar with `#nhg()`

- Probabilities simulated (Monte Carlo) based on `num-simulations` (default: `1000`)
- $\varepsilon_i \rightarrow$ single noise sample shown for illustration — not used by P_i

```
#nhg(
  input: "kraTa",
  candidates: ("[kra.Ta]", "[ka.Ta]",
"[ka.ra.Ta]"),
  constraints: ("Max", "Dep", "*Complex"),
  weights: (2.5, 1.8, 0.5),
  violations: (
    (0, 0, -1),
    (-1, 0, 0),
    (0, -1, 0),
  ),
  scale: 0.7,
)
```

	$w = 2.5$	$w = 1.8$	$w = 0.5$			
/kraθa/	Max	Dep	*Complex	h_i	ε_i	P_i
[kra.θa]	0	0	−1	−0.5	−0.47	0.778
[ka.θa]	−1	0	0	−2.5	−0.45	0.05
[ka.ra.θa]	0	−1	0	−1.8	−1.35	0.172

MaxEnt with `#maxent()`

- MaxEnt tableaux with automatic calculation and optional probability visualization

	$w = 2.5$	$w = 1.8$	$w = 0.5$			
/kraθa/	Max	Dep	*Complex	h_i	e^{-h_i}	P_i
[kra.θa]	0	0	1	0.5	0.607	0.71
[ka.θa]	1	0	0	2.5	0.082	0.096
[ka.ra.θa]	0	1	0	1.8	0.165	0.194

	$w = 2.5$	$w = 1.8$	$w = 0.5$			
/kraθa/	Max	Dep	*Complex	h_i	e^{-h_i}	P_i
[kra.θa]	0	0	1	0.5	0.607	0.71
[ka.θa]	1	0	0	2.5	0.082	0.096
[ka.ra.θa]	0	1	0	1.8	0.165	0.194

MaxEnt with `#maxent()`

- MaxEnt tableaux with automatic calculation and optional probability visualization

```
#maxent(
    input: "kraTa",
    candidates: ("[kra.Ta]", "[ka.Ta]", "[ka.ra.Ta]"),
    constraints: ("Max", "Dep", "*Complex"),
    weights: (2.5, 1.8, 0.5),
    violations: (
      (0, 0, 1),
      (1, 0, 0),
      (0, 1, 0),
    ),
    visualize: true, // ← visualization
    sort: true, // ← sort candidates by probability
)
```

MaxEnt with `#maxent()`

- You can also easily sort candidates by P_i with `sort: true` as of version `0.4.1`

	$w = 2.5$	$w = 1.8$	$w = 0.5$				
/kraθa/	Max	Dep	*Complex	h_i	e^{-h_i}	P_i	
[kra.θa]	0	0	1	0.5	0.607	0.71	
[ka.θa]	1	0	0	2.5	0.082	0.096	
[ka.ra.θa]	0	1	0	1.8	0.165	0.194	

	$w = 2.5$	$w = 1.8$	$w = 0.5$				
/kraθa/	Max	Dep	*Complex	h_i	e^{-h_i}	P_i	
[kra.θa]	0	0	1	0.5	0.607	0.71	
[ka.ra.θa]	0	1	0	1.8	0.165	0.194	
[ka.θa]	1	0	0	2.5	0.082	0.096	

Numbered examples with `#ex()`

- Phonology-friendly numbered examples: (1a) and (1b) are easy to reference
- Alignment is guaranteed given **table** structure; optional caption for table of contents

```
#show: ex-rules // ← this must be added to your doc
#ex(caption: "A phonology example")[
  #table(
    columns: 4, // ← where we may specify widths
    stroke: none,
    align: left,
    [#subex-label()<ex-anba>], [#ipa("/anba/")], [#a-
r], [#ipa("[amba]")],
    [#subex-label()<ex-anka>], [#ipa("/anka/")], [#a-
r], [#ipa("[aNka]")],
  )]
```

(1) a. /anba/ → [amba]
b. /anka/ → [aŋka]

FAQ

Common questions

1. Do I need to adopt Typst to take advantage of **phonokit**?
2. Can I completely replace LaTeX with Typst in 2026?
3. How about my `bib` references?
4. What *can't* I do with Typst?
5. What software do I need to use it?

FAQ & final thoughts

1. No. You can export outputs as `PNG` and use them in LaTeX, Word, etc. Pair it with `oxipng` for tiny file sizes. See workflow example in Garcia (2026, appendix).
2. That depends. Journals will take a while to accept `typ`, and very few people know Typst. But you don't have to choose: they're two useful tools/languages. If you work in phonology, you *could* probably use Typst 99% of the time. In syntax, LaTeX still offers more when it comes to trees.
3. They work with Typst. So your workflow will not be affected.
4. LaTeX is much older, so it has **many** more packages. What you can/can't do depends on what packages your workflow requires.
5. VS Code, Positron, NeoVim, etc. Use `tinymist` as your extension/plugin.

References

Booij, G. (2012). *The grammar of words: An introduction to linguistic morphology* (3rd ed.). Oxford University Press.

Carvalho, J. B. d. (2017). Deriving sonority from the structure, not the other way round: A Strict CV approach to consonant clusters. *The Linguistic Review*, *34*(4), 589–614.

Garcia, G. D. (2026,). *phonokit: a toolkit to create phonological representations in Typst*. Zenodo. https://doi.org/10.5281/zenodo.18434478

Goad, H. (2012). sC clusters are (almost always) coda-initial. *Linguistic Review*, *29*(3).

Parker, S. (2011). Sonority. In M. van Oostendorp, C. J. Ewen, E. Hume, & K. Rice (Eds.), *The Blackwell Companion to Phonology: The Blackwell Companion to Phonology* (pp. 1160–1184). Wiley Online Library. https://doi.org/10.1002/9781444335262.wbctp0049

Zsiga, E. C. (2024). *The sounds of language: An introduction to phonetics and phonology* (2nd ed.). John Wiley & Sons.