

EthioMorph: Technical Implementation of a Rule-Based Ge'ez Morphological Engine

Esubalew Chekol

College of Technology and Built Environment

Addis Ababa University

NLP Course Project

Abstract—Ge'ez verb conjugation follows strict patterns. Given a root like **፩፻፻**, every form (perfective, imperfective, jussive, imperative) can be computed by rearranging vowels and attaching affixes. No dictionary lookup, no statistical model—just rules.

This paper documents EthioMorph, a Python system I built to do exactly that. It handles all eight verb types that traditional Ge'ez grammar recognizes: **፩፻፻**, **፩፻፻**, **፩፻፻**, **፩፻፻**, **፩፻፻**, **፩፻፻**, **፩፻፻**, and **፩፻፻**. The core trick: Ethiopic Unicode assigns consecutive codepoints to vowel forms, so changing a vowel is just arithmetic. The rest is template matching and a few special rules for laryngeal consonants.

Index Terms—Ge'ez, morphological analysis, NLP, Unicode processing, Semitic morphology, rule-based systems

I. Introduction

I wanted to build a Ge'ez conjugator that works from first principles. Not a lookup table with thousands of pre-stored forms, and not a neural network trained on text. Just the rules, written in code.

Why is this even possible? Because Ge'ez is regular. The same root (**፩፻፻**, meaning “kill”) always conjugates the same way. There are eight verb types and maybe a dozen special cases for guttural consonants. Once you encode those rules, you can generate or analyze any verb.

A. What the System Does

Ge'ez verbs have two parts: a consonant root (usually 3 letters) and a vowel pattern. The root carries meaning, the pattern carries grammar.

Take **q-t-l** (**፩፻፻**, “kill”). Same three consonants, different vowels:

q-t-l + [1-1-1] → **፩፻፻** (past: “he killed”)

q-t-l + [yi-1-6-6] → **፩፻፻** (present: “he kills”)

EthioMorph does this in both directions:

Analysis: **፩፻፻** → root: **፩፻፻**, tense: imperfective, subject: 3rd plural masculine

Generation: (**፩፻፻**, imperfective, 3PM) → **፩፻፻**

II. Unicode Mathematics for Ethiopic

A. The Ethiopic Block Structure

The Ethiopic Unicode block (U+1200–U+137F) organizes characters in a systematic pattern. Each consonant has 7 forms representing vowel variations:

TABLE I
Unicode Pattern for Consonant **፩** (qäf)

Order	Unicode	Character	Vowel
1	U+1240	፩	ä (schwa)
2	U+1241	፪	u
3	U+1242	፫	i
4	U+1243	፬	a
5	U+1244	፭	e
6	U+1245	፮	(none)
7	U+1246	፯	o

B. Vowel Arithmetic

Since vowel forms are consecutive in Unicode, you can do math on them:

$$\text{order}(c) = ((c - \text{base_offset}) \bmod 8) + 1 \quad (1)$$

$$\text{base}(c) = c - (\text{order}(c) - 1) \quad (2)$$

To change a vowel:

$$\text{revowelize}(\text{base}, \text{order}) = \text{base} + (\text{order} - 1) \quad (3)$$

Implementation:

```
1 def get_vowel_order(char):
2     """Extract vowel order (1-7) from character."""
3     return ORDER_MAP.get(char, 1)
4
5 def devowelize(char):
6     """Return base consonant (1st order form)."""
7     return DEVOWELIZATION_MAP.get(char, char)
8
9 def get_char_by_order(base, order):
10    """Apply vowel order to base consonant."""
11    return REVOWELIZATION_MAP.get((base, order), base)
```

Example transformation:

$$\text{devowelize}(\text{፩}) = \text{፩}$$

$$\text{get_vowel_order}(\text{፩}) = 4$$

$$\text{get_char_by_order}(\text{፩}, 6) = \text{፩}$$

III. The C1-C2-C3 Radical System

A. Naming the Radicals

I use C1, C2, C3 to refer to the consonants:

- **C1:** First radical (consonant 1)
 - **C2:** Second radical
 - **C3:** Third radical
 - **C4:** Fourth radical (quadriliterals only)

For $\Phi + \Lambda$: C1= Φ , C2= Λ , C3=

B. How Templates Work

Each conjugation form is just a vowel pattern plus optional prefix/suffix:

```

1 # Perfective 3rd singular masculine for
2     Type A
3 template = {
4     "vowel_map": {"C1": 1, "C2": 1, "C3": 1},
5     "prefix": "",
6     "suffix": ""
7 }
8 # Result: C1(1st) + C2(1st) + C3(1st) = qä-
9     tä-lä

1 # Imperfective 3rd singular masculine
2 template = {
3     "vowel_map": {"C1": 1, "C2": 6, "C3": 6},
4     "prefix": "yi",
5     "suffix": ""
6 }
7 # Result: yi + C1(1st) + C2(6th) + C3(6th)
8     = yi-qä-t-l

```

C. Putting It Together

Algorithm 1 Generate a conjugated word

Require: root (string), tense, subject, verb_type

Ensure: conjugated word (string)

```

1: template ← TEMPLATES[verb_type][tense][subject]
2: vowel_map ← template["vowel_map"]
3: radicals ← extract_radicals(root)
4: result ← template["prefix"]
5: for each position in ["C1", "C2", "C3"] do
6:   base ← devowelize(radicals[position])
7:   order ← vowel_map[position]
8:   result ← result + get_char_by_order(base, order)
9: end for
10: result ← result + template["suffix"]
11: result ← apply_suffix_fusion(result)
12: return result

```

IV. Suffix Fusion Rules

A. When Suffixes Collide

When you attach a vowel suffix to a consonant-only ending (6th order), you can't just concatenate them:

$$C_{6th} + V_{\text{suffix}} \rightarrow C_{\text{order}(V)} \quad (4)$$

Example: $\Phi\tau\alpha + \kappa$ ("they") $\neq \Phi\tau\alpha\kappa$

Instead: $\Phi\bar{\tau}\Lambda + h \rightarrow \Phi\bar{\tau}\Lambda$

B. Fusion Algorithm

```

1 def apply_suffix_fusion(stem, suffix):
2     """Fuse 6th-order final + vowel suffix.
3         """
4
5     if not suffix:
6         return stem
7
8     last_char = stem[-1]
9     last_order = get_vowel_order(last_char)
10
11    # Check if stem ends in 6th order (consonant)
12    if last_order != 6:
13        return stem + suffix
14
15    # Map suffix to target order
16    suffix_vowel_map = {
17        '\u12A1': 2,    # u-vowel suffix
18        '\u12A0': 4,    # a-vowel suffix
19        '\u12A5': 5,    # e-vowel suffix
20    }
21
22    first_suffix_char = suffix[0]
23    if first_suffix_char in
24        suffix_vowel_map:
25        target_order = suffix_vowel_map[
26            first_suffix_char]
27        base = devowelize(last_char)
28        fused = get_char_by_order(base,
29            target_order)
30    return stem[:-1] + fused + suffix
31        [1:]
32
33    return stem + suffix

```

V. Verb Home Detection Algorithm

A. The Eight Canonical Verb Types

In Classical Ge'ez grammar, verbs are classified into “homes” (**አት**) named after their canonical representative. EthioMorph implements all eight standard verb types:

TABLE II
The Eight Canonical Ge'ez Verb Types (Homes)

#	Type	Head	Name	Description
1	ቀተለ		Type A	Strong triradical (C1 = 1st order)
2	ቃይና		Type B	Geminate (C2 doubles in conjugation)
3	በረከ		Type C	Long vowel (C1 = 4th order እ-ለብ)
4	መመራ		Type C-O	O-initial (C1 = 7th order አ-ለብ)
5	ለሆነ		Weak-Final	Final radical is ባ (weak)
6	ከሁለ		Laryngeal	Contains laryngeal (v) radical
7	ማስከራከ		Quadriliteral	Four consonant radicals
8	ተ-ጻዢለ		T-Quad	Quadriliteral with ተ- prefix

B. How to Classify a Verb

Each verb type uses different conjugation templates, so you need to figure out the type first. The algorithm checks:

- How many radicals? (3 = triradical, 4 = quadrilateral)
- What vowel does C1 have? (4th order = Type C, 7th order = Type C-O)
- Any laryngeals (**v**, **ħ**, **ّ**, **ك**, **و**) or weak consonants (**ء**, **়**)?
- Does C2 double? (That's Type B)

TABLE III
Algorithmic Classification Rules

Condition	Type	Example
$ \text{radicals} \geq 4$	Quadrilateral	مڭڭڭڭ , تۇۇۇۇ
C1 order = 7	Type C-O	مۇۇۇ
C1 order = 4	Type C	ئۇۇ
$C3 \in \{\text{weak}\}$	Weak-Final	لۇۇ
$C2 \in \{\text{laryngeal}\}$	Laryngeal	ئۇۇ
C2 geminated	Type B	فڭڭ
Default	Type A	فۇۇ

C. Detection Algorithm

D. Feature Detection

Some roots have special properties that affect conjugation:

```

1 LARYNGEALS = {'h', 'H', 'x', 'ا', 'أ'} # Gutturals
2 WEAK_CONSONANTS = {'w', 'y'}
3
4 def detect_features(root, radicals):
5     features = {}
6
7     # Check for laryngeal consonants
8     for rad in radicals:
9         base = devowelize(rad)
10        if base in LARYGEAL_BASES:
11            features['has_laryngeal'] =
12                True
13            break
14
15        # Check for hollow verb (weak C2)
16        if len(radicals) >= 2:
17            c2_base = devowelize(radicals[1])
18            if c2_base in WEAK_BASES:
19                features['is_hollow'] = True
20
21        # Check for weak initial
22        c1_base = devowelize(radicals[0])
23        if c1_base in WEAK_BASES:
24            features['weak_initial'] = True
25
26    return features

```

VI. Laryngeal Vowel Shift Rules

A. Gutturals Are Picky

Laryngeal consonants (**v**, **ħ**, **ّ**, **ك**, **و**) don't accept certain vowels. When you try to apply a 1st order vowel to a laryngeal, it shifts:

Algorithm 2 Figure out which verb type this is

```

Require: root (string)
Ensure: (type, features)
1: radicals ← extract_consonants(root)
2: features ← {}
3: // Check radical count first
4: if len(radicals) ≥ 4 then
5:   if root starts with ت then
6:     return ("type_tanbala", features) {تۇۇۇۇ}
7:   else
8:     return ("type_mahraka", features) {مۇۇۇۇ}
9:   end if
10: end if
11: // Check C1 vowel order
12: c1_order ← get_vowel_order(root[0])
13: if c1_order == 7 then
14:   return ("type_c_o", features) {مۇۇۇۇ}
15: else if c1_order == 4 then
16:   return ("type_c", features) {ئۇۇ}
17: end if
18: // Check for weak final radical
19: if devowelize(root[2]) ∈ WEAK_CONSONANTS then
20:   features["weak_final"] ← True
21:   return ("type_sesaya", features) {لۇۇ}
22: end if
23: // Check for laryngeal radicals
24: for each radical in radicals do
25:   if devowelize(radical) ∈ LARYNGEALS then
26:     features["has_laryngeal"] ← True
27:     return ("type_kahla", features) {ئۇۇ}
28:   end if
29: end for
30: // Check for gemination (Type B)
31: if is_geminate_pattern(root) then
32:   return ("type_b", features) {فڭڭ}
33: end if
34: // Default: Type A
35: return ("type_a", features) {فۇۇ}

```

$$\text{Laryngeal} + 1\text{st order} \rightarrow \text{Laryngeal} + 4\text{th order} \quad (5)$$

B. Implementation

```

1 def apply_laryngeal_rules(char, position,
2                             features):
3     """Shift vowels near laryngeal
4             consonants."""
5     if not features.get('has_laryngeal'):
6         return char
7
8     base = devowelize(char)
9     order = get_vowel_order(char)
10
11    # Laryngeals prefer 4th order in
12        jussive
13    if is_laryngeal(base) and order == 1:
14
15
16
17
18
19
10

```

```

11     return get_char_by_order(base, 4)
12
13     return char

```

VII. Stemmer: Going Backwards

The Stemmer takes a conjugated word and figures out its root. It's the reverse of generation.

A. The Steps

- 1) Normalize spelling (Ge'ez has some redundant characters)
- 2) Strip prefixes (**ፋ**, **ጥ**, **ክ**, etc.)
- 3) Strip suffixes (**ኩ**, **ኩ**, etc.)
- 4) Get the consonant skeleton
- 5) Restore any weak consonants that disappeared
- 6) Figure out tense/person from what you stripped
- 7) Classify the verb type

B. Checking Prefixes

You have to check longer prefixes first, or you'll make wrong matches:

```

1 PREFIXES = [
2     # Stem IV (Causative-Passive) - check
3         # first
4     'yaste', 'taste', 'naste', 'laste',
5     # Stem III (Causative)
6     'yas', 'tas', 'nas', 'las',
7     'ya', 'ta', 'na', 'a',
8     # Stem II (Passive)
9     'yit', 'tit', 'te',
10    # Stem I (Basic)
11    'yi', 'ti', 'ni', 'li'
12]
13 # Sorted by length descending
PREFIXES.sort(key=len, reverse=True)

```

C. Pattern Identification

```

1 def identify_verb_pattern(stem,
2     stripped_prefixes):
3     """Determine grammatical pattern from
4         affixes."""
5
6     # Check stem markers in priority order
7     if 'aste' in stripped_prefixes:
8         return {'stem': 4, 'name': 'Causative-Passive'}
9
10    if 'as' in stripped_prefixes or
11        has_causative_vowel(stem):
12        return {'stem': 3, 'name': 'Causative'}
13
14    if 'te' in stripped_prefixes or 'yit'
15        in stripped_prefixes:
16        return {'stem': 2, 'name': 'Passive'}
17
18    return {'stem': 1, 'name': 'Basic'}

```

VIII. Data Structures

A. Templates JSON

```

1 {
2     "type_a": {
3         "perfective": {
4             "3sm": {
5                 "vowel_map": {"C1": 1, "C2": 1, "C3": 1},
6                 "prefix": "",
7                 "suffix": ""
8             },
9             "3sf": {
10                 "vowel_map": {"C1": 1, "C2": 1, "C3": 4},
11                 "prefix": "",
12                 "suffix": "t"
13             }
14         // ... 10 person forms
15     },
16     "imperfective": {
17         "3sm": {
18             "vowel_map": {"C1": 1, "C2": 6, "C3": 6},
19             "prefix": "yi",
20             "suffix": ""
21         }
22         // ...
23     }
24     // jussive, imperative, derived forms
25 }
26

```

B. Lookup Maps

I precompute three dictionaries so character lookups are instant:

```

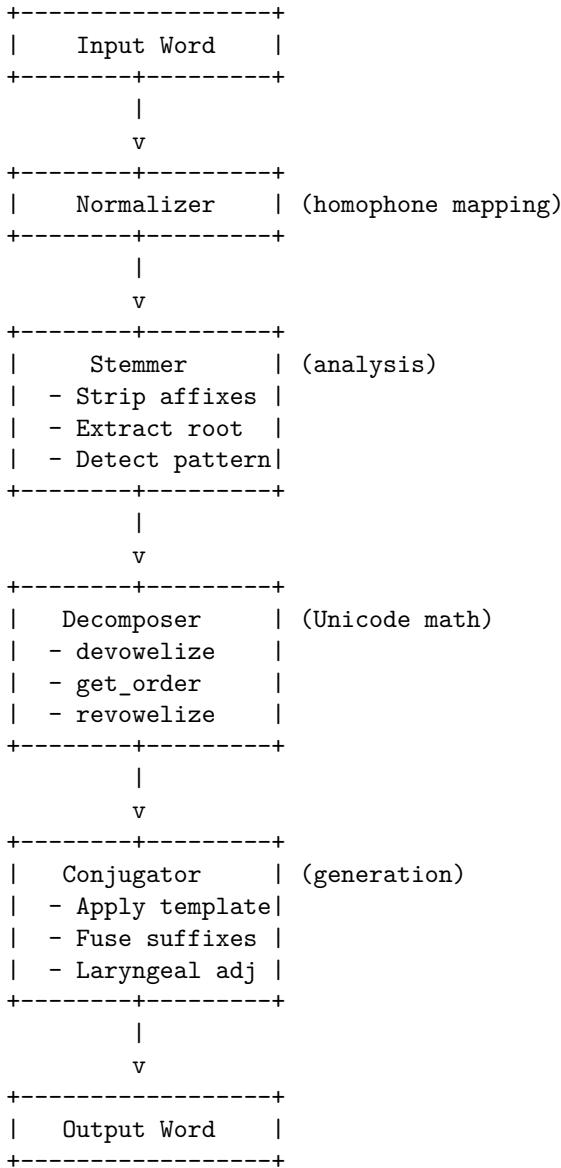
1 # Character -> Base consonant
2 DEVOWELIZATION_MAP = {
3     '\u1240': '\u1240', # ge -> ge (
4         already base)
5     '\u1241': '\u1240', # qu -> ge
6     '\u1242': '\u1240', # qi -> ge
7     # ... all 300+ characters
8 }
9
10 # Character -> Vowel order (1-7)
11 ORDER_MAP = {
12     '\u1240': 1, # ge = 1st order
13     '\u1241': 2, # qu = 2nd order
14     # ...
15 }
16
17 # (Base, Order) -> Character
18 REVOWELIZATION_MAP = {
19     ('\u1240', 1): '\u1240',
20     ('\u1240', 2): '\u1241',
21     ('\u1240', 3): '\u1242',
22     # ...
23 }

```

IX. System Architecture

X. API Usage

A. Analysis Endpoint



```

2  {
3   "root": "qatala"
4 }
5
6 Response:
7 {
8   "perfective": {
9     "3sm": "qatala",
10    "3sf": "qatalat",
11    ...
12  },
13  "imperfective": {...},
14  "jussive": {...}
15 }

```

XI. What I Learned

Building this taught me that Ge'ez morphology is more regular than I expected. The Unicode trick—that vowel forms are consecutive codepoints—turns what seems like a complex linguistic operation into basic arithmetic. Most of the code is just applying templates and handling edge cases for gutturals.

The system isn't complete. It handles the eight main verb types and basic stems, but there are derived forms and obscure patterns I haven't implemented. Still, it works well enough to conjugate most verbs you'd encounter in classical texts.

Code: github.com/esubalew/EthioMorph

Fig. 1. How a word flows through the system

```

1 POST /api/xray
2 {
3   "word": "yiqatlu"
4 }
5
6 Response:
7 {
8   "root": "qatala",
9   "pattern": "imperfective",
10  "stem": 1,
11  "person": "3pm",
12  "verb_type": "type_a"
13 }

```

B. Generation Endpoint

```
1 POST /api/morph
```