The isiZulu verbalisation algorithms: design and documentation

C. Maria Keet

Department of Computer Science, University of Cape Town, South Africa
mkeet@cs.uct.ac.za

May 27, 2018

Abstract

Automatically generating text in isiZulu—the largest language by first language speakers in South Africa—has been investigated over the past few years. This was done in an incremental fashion, covering one feature at a time. The principal three components are generating plurals from a noun in the singular, the main axiom types in an ontology, such as subsumption of named classes and existential quantification, and the phonological conditioning that is required in certain cases. This document lists the core algorithms and contains brief explanatory descriptions, which serve as implementation-independent documentation of the code and to describe those algorithms not included in the respective papers. The code is available as a downloadable zip file from the project website at http://www.meteck.org/geni/, with as cut-off date the state in February 2018.

Keywords: Natural Language Generation, Controlled Natural Language, Pluralisation, Phonological Conditioning, isiZulu, OWL

Contents

1 Pluralisation of nouns 2
2 Verbalisation algorithms for (almost) $ALC$ 4
3 Algorithms for basic part-whole relations 8
   3.1 Whole-part reading direction 8
   3.2 Part-whole reading direction 12
4 Phonological conditioning 15
5 Architecture of the verbaliser 19
1 Pluralisation of nouns

The algorithm to pluralise nouns in the singular included here (Algorithm 1) is slightly changed cf. the one described in [1] and implemented in its related supplementary material, for there it was a stand-alone file and read in the test sets, whereas here that ‘wrapping’ is omitted.

Algorithm 1 Pluralise isiZulu noun (full version)

1: procedure PLURALISE(n)
2: input: noun in singular  
   \{note: n noun, nc noun class, p plural\}
3: if n in exceptionList then  
   \{check exceptions list\}
4: p ← plural exception  
5: else
6: nc ← getNC(n)  
   \{lookup noun class of noun n\}
7: \{pluralise compound nouns\}
8: if ' ' ∈ n and not endswith(nc, m) then
9:   main, rest ← split(n)  
   \{to pluralise main noun and modify the modifier word (rest)\}
10: switch
11:   case nc == 1 and rest[0] /∈ \{a, e, i, o, u\}
12:     rest' = b + rest[1:]
13:   case nc == 1 and rest[0] ∈ \{a, e, i, o, u\}
14:     rest' = aba + rest[1:]
15:   case nc == 3a
16:     rest' = aba + rest[1:]
17:   case (nc == 9 or nc == 7) and rest[0] /∈ \{a, e, i, o, u\}
18:     rest' = z + rest[1:]
19:   case (nc == 9 or nc == 7) and rest[0] ∈ \{a, e, i, o, u\}
20:     rest' = rest[0] + zi + rest[3:]
21: break  
   \{cases for other nc are yet to be investigated\}
22: end switch
23: p' ← pluralise(main)
24: p ← p' + ' ' + rest'
25: else
26: \{pluralise the regular nouns\}
27: switch
28:   case (startswith(n, um) or startswith(n, UM)) and n[2] != u and nc == 1
29:     p = aba + n[2:]
30:   case (startswith(n, um) or startswith(n, UM)) and n[2] ∈ \{a, e, i, o\}
31:     p = ab + n[2:]
32:   case startswith(n, umu) and nc == 1
33:     p = aba + n[3:]
34:   case startswith(n, u) and nc == 1a or nc == 3a
35:     p = o + n[1:]
36:   case startswith(n, um) and n[2] != u and nc == 3
37:     p = imi + n[2:]
38:   case startswith(n, umu) and nc == 3
39:     p = imi + n[3:]
40: end switch

2
case startswith(n, i) and n[0:2] != ili and (nc == 5 or nc == 9a)
  p = ama + n[1:]
end if

case startswith(n, ili) and nc == 5
  p = ama + n[3:]
end if

case startswith(n, isi) and nc == 7
  p =izi + n[3:]
end if

case startswith(n, is) and nc == 7 and n[2] ∈ {a, e, o, u}
  p = iz + n[2:]
end if

case startswith(n, im) and nc == 9
  p = izi + n[1:]
end if

case (startswith(n, in) or startswith(n, iM)) and nc == 9
  p = izin + n[2:]
end if

case startswith(n, ulu) and nc == 11
  p = izi + n[3:]
end if

case startswith(n, u) n[1:2] != lu and nc == 11
  p = izi + n[1:]
end if

case (startswith(n, uku) or startswith(n, uk)) and (nc == 15 or nc == 17)
  {nc15 and nc17 don’t pluralise}
end if

case endswith(nc, m)
  p ← n
  {mass nouns don’t pluralise}
end if

case nc ∈ {2, 4, 6, 8, 2a, 10}
  {noun exists only in plural form}
  p = n
break
end if
end if

return p
end procedure
2 Verbalisation algorithms for (almost) ALC

The following list of algorithms are included in this document, which were first described in [2, 3] and more comprehensively in [6]:

- Simple taxonomic subsumption, i.e., named class subsumption of the axiom type \( C \sqsubseteq D \), in Algorithm 2;
- Simple existential quantification with named classes, i.e., of the axiom type \( C \sqsubseteq \exists R.D \), in Algorithm 3;
- Negation in an axiom, covering both the axiom types \( C \sqsubseteq \neg D \) and \( C \sqsubseteq \neg \exists R.D \), in Algorithm 4. This algorithm has been updated with vowel-commencing verb roots cf. the one presented in [6].

Algorithm 3 and Algorithm 4 have functions to look up things from a list. They are the ‘lookup tables’ for noun classes, and their corresponding quantitative, relative, and (negative) subject concords, which are included in Table 1 for easy readable reference.

Table 1: Zulu noun classes with examples and a selection of concords. NC: Noun class; PRE: prefix; QC: quantitative concord; RC: relative concord; SC: subject concord; NEG SC: negative subject concord; PC: possessive concord. Updated cf. the tables in [1, 5, 6] (deviant cases of prefixes not included).

<table>
<thead>
<tr>
<th>NC</th>
<th>Full PRE</th>
<th>QC (⊇)</th>
<th>RC</th>
<th>QC (⊇)</th>
<th>SC</th>
<th>NEG SC</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>um(u)-</td>
<td>wonke</td>
<td>o-</td>
<td>ye-</td>
<td>u-</td>
<td>aka-</td>
<td>wa-</td>
</tr>
<tr>
<td>2</td>
<td>aba-</td>
<td>wonke</td>
<td>o-</td>
<td>ye-</td>
<td>u-</td>
<td>aka-</td>
<td>ba-</td>
</tr>
<tr>
<td>2a</td>
<td>o-</td>
<td>wonke</td>
<td>o-</td>
<td>ye-</td>
<td>u-</td>
<td>aka-</td>
<td>ba-</td>
</tr>
<tr>
<td>3a</td>
<td>u-</td>
<td>wonke</td>
<td>o-</td>
<td>ye-</td>
<td>u-</td>
<td>aka-</td>
<td>wa-</td>
</tr>
<tr>
<td>2a</td>
<td>o-</td>
<td>wonke</td>
<td>o-</td>
<td>ye-</td>
<td>u-</td>
<td>aka-</td>
<td>ba-</td>
</tr>
<tr>
<td>3</td>
<td>um(u)-</td>
<td>wonke</td>
<td>o-</td>
<td>ye-</td>
<td>u-</td>
<td>aka-</td>
<td>ba-</td>
</tr>
<tr>
<td>4</td>
<td>imi-</td>
<td>yonke</td>
<td>e-</td>
<td>yo-</td>
<td>u-</td>
<td>awu-</td>
<td>wa-</td>
</tr>
<tr>
<td>5</td>
<td>i(l)-</td>
<td>lonke</td>
<td>e-</td>
<td>yo-</td>
<td>u-</td>
<td>awu-</td>
<td>wa-</td>
</tr>
<tr>
<td>6</td>
<td>ama-</td>
<td>lonke</td>
<td>e-</td>
<td>yo-</td>
<td>u-</td>
<td>awu-</td>
<td>wa-</td>
</tr>
<tr>
<td>7</td>
<td>isi-</td>
<td>sonke</td>
<td>e-</td>
<td>yo-</td>
<td>i-</td>
<td>ayi-</td>
<td>ya-</td>
</tr>
<tr>
<td>8</td>
<td>izi-</td>
<td>sonke</td>
<td>e-</td>
<td>yo-</td>
<td>i-</td>
<td>ayi-</td>
<td>ya-</td>
</tr>
<tr>
<td>9a</td>
<td>i-</td>
<td>onke</td>
<td>e-</td>
<td>yo-</td>
<td>a-</td>
<td>awa-</td>
<td>a-</td>
</tr>
<tr>
<td>6</td>
<td>ama-</td>
<td>onke</td>
<td>e-</td>
<td>yo-</td>
<td>a-</td>
<td>awa-</td>
<td>a-</td>
</tr>
<tr>
<td>9</td>
<td>i(n)-, i(m)-</td>
<td>onke</td>
<td>e-</td>
<td>yo-</td>
<td>i-</td>
<td>ayi-</td>
<td>ya-</td>
</tr>
<tr>
<td>10</td>
<td>i(z)-, i(zi)-</td>
<td>onke</td>
<td>e-</td>
<td>yo-</td>
<td>i-</td>
<td>ayi-</td>
<td>ya-</td>
</tr>
<tr>
<td>11</td>
<td>u(lu)-</td>
<td>lonke</td>
<td>e-</td>
<td>zu-</td>
<td>l-</td>
<td>alu-</td>
<td>lwa-</td>
</tr>
<tr>
<td>10</td>
<td>i(zi)-, i(zi)-</td>
<td>lonke</td>
<td>e-</td>
<td>zu-</td>
<td>l-</td>
<td>alu-</td>
<td>lwa-</td>
</tr>
<tr>
<td>14</td>
<td>ubu-</td>
<td>onke</td>
<td>e-</td>
<td>zu-</td>
<td>l-</td>
<td>alu-</td>
<td>lwa-</td>
</tr>
<tr>
<td>15</td>
<td>uku-</td>
<td>konke</td>
<td>e-</td>
<td>zu-</td>
<td>l-</td>
<td>alu-</td>
<td>lwa-</td>
</tr>
<tr>
<td>17</td>
<td>ku-</td>
<td>lonke</td>
<td>o-</td>
<td>lo-</td>
<td>l-</td>
<td>kwa-</td>
<td>kwa-</td>
</tr>
</tbody>
</table>

Further, note that these algorithms require pluralisation of the head noun, whose algorithm is included in Section 1 (Algorithm 1).

There are specific cases with the part-whole relations, which are described in Section 3. Both they and ‘regular irregular’ verbs require phonological conditioning, which is described in Section 4.
Algorithm 2 (TaxSubs) Verbalisation of taxonomic subsumption, named classes ($C \sqsubseteq D$).

**Require:** $C$ set of classes, language $\mathcal{L}$ with $\sqsubseteq$ for subsumption and $\neg$ for negation; variables: $A$ axiom, $NC_i$ nounclass, $c_1, c_2 \in C$, $a_1$ term, $a_2$ letter; functions: $\text{getFirstClass}(A)$, $\text{getSecondClass}(A)$, $\text{getNC}(C)$, $\text{checkNegation}(A)$, $\text{getFirstChar}(C)$.

**Require:** axiom $A$ with a $\sqsubseteq$ has been retrieved and named classes on the lhs and rhs

1: $c_1 \leftarrow \text{getFirstClass}(A)$ \{get subclass\}
2: $c_2 \leftarrow \text{getSecondClass}(A)$ \{get superclass\}
3: $NC_1 \leftarrow \text{getNC}(c_1)$ \{determine noun class by augment and prefix or dictionary\}
4: $NC_2 \leftarrow \text{getNC}(c_2)$ \{determine noun class by augment and prefix or dictionary\}
5: if $\text{checkNegation}(A) == \text{true}$ then
6: \{use negation (Algorithm 4)\}
7: else
8: $a_2 \leftarrow \text{getFirstChar}(c_2)$ \{retrieve first letter of $c_2$\}
9: switch
10: case $a_2 = \text{‘i’}$ then
11: \hspace{1em} $\text{RESULT} \leftarrow \text{‘}c_1 \text{ } y c_2.$’ \{verbalise as taxonomic subsumption with $y$\}
12: case $a_2 = \{\text{‘a’, ‘o’, ‘u’}\}$ then
13: \hspace{1em} $\text{RESULT} \leftarrow \text{‘}c_1 \text{ } ng c_2.$’ \{verbalise as taxonomic subsumption with $ng$\}
14: case $a_2 \notin \{\text{‘a’, ‘i’, ‘o’, ‘u’}\}$ then
15: \hspace{1em} $\text{RESULT} \leftarrow \text{‘}this \text{ is not a well-formed isiZulu noun.’}$
16: end switch
17: end if
18: return $\text{RESULT}$
Algorithm 3 (AllSome) Verbalisation of “all-some” axiom type (C ⊑ ∃R.D)

Require: C set of classes, language L with ⊑ for subsumption and ∃ for existential quantification; variables: A axiom, NC_i noun class, c_1, c_2 ∈ C, o ∈ R, a_1 a term; r_2, q_2 concords;
functions: getFirstClass(A), getSecondClass(A), getNC(C), getRC(NC_i), getQC(NC_i), getV SofOP(o).

Require: axiom A with a ⊑ has been retrieved and an ∃ on the rhs of the inclusion
1: c_1 ← getFirstClass(A) \{get subclass\}
2: c_2 ← getSecondClass(A) \{get superclass\}
3: o ← getObjProp(A) \{get object property\}
4: v ← getV SofOP(o) \{get verb stem of object property\}
5: NC_1 ← getNC(c_1) \{determine noun class by augment and prefix or dictionary\}
6: NC_2 ← getNC(c_2) \{determine noun class by augment and prefix or dictionary\}
7: NC'_1 ← lookup plural nounclass of NC_1 \{from known list\}
8: c'_1 ← pluralise(c_1, NC'_1) \{call algorithm pluralise to generate a plural from o\}
9: a_1 ← lookup quantitative concord for NC'_1 \{from quantitative concord (QC(all)) list\}
10: r_2 ← getRC(NC_2) \{get relative concord for c_2 from the QCdwa-list\}
11: q_2 ← getQC(NC_2) \{get quantitative concord for c_2 from the QCdwa-list\}
12: if checkNegation(A) == true then
13: \{use negation (Algorithm 4)\}
14: else
15: if o annotated with present tense then
16: conj nc1 ← lookup SC of NC'_1 \{from known SC list\}
17: o' ← conj nc1 v \{generate conjugated verb\}
18: Result ← ‘a_1 c'_1 o' a_2 r_2 q_2 dwa.’ \{verbalise the axiom\}
19: else
20: Result ← ‘passive voice and inverses are not supported yet.’
21: end if
22: end if
23: return Result
Algorithm 4 (Negation) Verbalisation of negation in an axiom, as disjointness or negated object property (i.e., axioms of type $C \sqsubseteq \neg D$ and $C \sqsubseteq \neg \exists R.D$).

**Require:** $C$ set of classes, language $L$ with $\sqsubseteq$ for subsumption and $\neg$ for negation; variables:
- $A$ axiom, $NC_i$ noun class, $c_1, c_2 \in C$, $a_1$ term, $a_2$ letter and $n, p$ are concords, $v$ verb stem;
- functions: $\text{checkNegation}(A)$, $\text{getNSC}(NC_i)$, $\text{getPNC}(NC_i)$.

**Require:** $\text{checkNegation}(A) == \text{true}$

1: if negation directly preceded by $\sqsubseteq$ and directly followed by $c_2$ then
2: \hspace{1em} $NC_i' \leftarrow$ lookup plural nounclass of $NC_i$ \hspace{1em} \{ from known list \}
3: \hspace{1em} $c_1' \leftarrow \text{pluralise}(c_1, NC_i')$ \hspace{1em} \{ call algorithm pluralise to generate a plural from $o$ \}
4: \hspace{1em} $a_1 \leftarrow$ lookup quantitative concord for $NC_i'$ \hspace{1em} \{ from quantitative concord (QC(all)) list \}
5: \hspace{1em} $n \leftarrow \text{getNSC}(NC_i')$ \hspace{1em} \{ get negative subject concord for $c_1'$ \}
6: \hspace{1em} $p \leftarrow \text{getPNC}(NC_2)$ \hspace{1em} \{ get pronomial for $c_2$ \}
7: \hspace{1em} Result $\leftarrow \text{'}a_1 \ c_1' \ np \ c_2\text{'}.$ \hspace{1em} \{ verbalise the disjointness ($a_1$ is QC(all)) \}
8: else if negation in front of OP then
9: \hspace{1em} $v' \leftarrow$ remove final vowel of $v$ \hspace{1em} \{ i.e., obtain the (possibly extended) verb root \}
10: \hspace{1em} $n \leftarrow \text{getNSC}(NC_i')$ \hspace{1em} \{ get negative subject concord for $c_1'$ \}
11: \hspace{1em} if $v' \in \{a, e, i, o, u, \}$ then
12: \hspace{2em} negv $\leftarrow \text{phonCondNegSc}(v', n)$ \hspace{1em} \{ verbalise the axiom \}
13: \hspace{2em} else
14: \hspace{3em} negv $\leftarrow n + v'$ \hspace{1em} \{ negation in front of $c_2$ and $A$ contains an OP \}
15: \hspace{1em} end if
16: \hspace{1em} Result $\leftarrow \text{'}a_1 \ c_1' \ negvi \ c_2 \ r_2q_2\text{'}dwa.$ \hspace{1em} \{ verbalisation of this class negation is not supported yet. \}
17: else
18: \hspace{1em} Result $\leftarrow \text{'}\text{verbalisation of this class negation is not supported yet.}.$ \hspace{1em} \{ negation in front of $c_2$ and $A$ contains an OP \}
19: end if
20: return Result
3 Algorithms for basic part-whole relations

The algorithms are here presented as functions that integrate with the other algorithms presented in the preceding sections, in the sense that only the “all some” the axiom type is considered, i.e., \( C \sqsubseteq \exists R. D \), where in these cases, \( R \) is the ‘has part’ or the ‘part of’ reading direction. For rationale and descriptions of the verbalisation patterns, see the corresponding paper [5]; an informal summary of the part-whole relations is shown in Figure 1. The ‘has part’ direction algorithm (Algorithm 5) was first published in [5].

Figure 1: Preliminary taxonomy based on the verbalisation patterns in [5] (source: [4]).

3.1 Whole-part reading direction

**Algorithm 5** Determine the verbalisation of basic whole-part in an axiom. This covers the structural, involvement, containment, membership, part-subquantities, and participation whole-part relations

**Require:** \( C \) set of classes, language \( L \), \( \sqsubseteq \) for subsumption, \( \exists \) for existential quantification; variables: \( A \) axiom, \( NC_i \) noun class, \( w, p \in C, o \in R, a_w \) a term; \( r_p, q_p \) concords;

**Require:** axiom of the form \( W \sqsubseteq \exists wp. P \) has been retrieved for verbalisation

1: \( w \leftarrow \text{getFirstClass}(A) \) \{get whole\}
2: \( p \leftarrow \text{getSecondClass}(A) \) \{get part\}
3: \( wp \leftarrow \text{getObjProp}(A) \) \{get \( wp \) type (‘default’ parthood here)\}
4: \( NC_w \leftarrow \text{getNC}(w) \) \{obtain noun class whole\}
5: \( NC_p \leftarrow \text{getNC}(p) \) \{obtain noun class part\}
6: \( wp_{pl} \leftarrow \text{pluralise}(w, NC_w) \) \{generate plural, using the pluraliser algorithm\}
7: \( NC'_{w} \leftarrow \text{getPlNC}(NC_{w}) \) \{obtain plural NC, from known list\}
8: \( a_w \leftarrow \text{getQCAll}(NC'_{w}) \) \{obtain quantitative concord (QC(all))\}
9: \( s_w \leftarrow \text{getSC}(NC'_{w}) \) \{obtain subject concord\}
10: \( conjp \leftarrow \text{phonoCondition}(\text{‘na’}, p) \) \{prefix P with the CONJ, phonologically conditioned\}
11: \( r_p \leftarrow \text{getRC}(NC_p) \) \{obtain relative conc. for \( p \)\}
12: \( q_p \leftarrow \text{getQC}(NC_p) \) \{obtain quant. concord for \( p \) from the QC (exists)-list\}
13: \( \text{RESULT} \leftarrow \text{‘} a_w w_{pl} s_w conjp r_p q_p dwa \text{’} \) \{verbalise the simple axiom\}
14: \textbf{return RESULT}

Because there is quite some duplication, like fetching the classes, pluralising, and adding the quantitative concords, we put this now in a separate algorithm, \( \text{commonFunctWP} \), being Algorithm 6, that will be called by all the other functions. In some cases, it fetches a bit more than strictly needed (e.g., an RC and QC too much), but it saves a lot of duplication in the presentation here, and it’s not computationally costly (linear, with a small list). The solid portions deviate from this, due to mostly dealing with a noun phrase (e.g., ‘sample of blood’), so it is written in full there (Algorithm 10).
Algorithm 6 Common functions for wp verbalisation, commonFunctWP.

Require: \(C\) set of classes, language \(L\), \(\sqsubseteq\) for subsumption, \(\exists\) for existential quantification; variables: \(A\) axiom, \(NC_i\) noun class, \(w, p \in C\), \(o \in R\), \(a_w\) a term; \(r_p, q_p\) concords;

Require: axiom of the form \(W \sqsubseteq \exists wp.P\) has been retrieved for verbalisation

1: \(w \leftarrow \text{getFirstClass}(A)\) \{get whole\}
2: \(p \leftarrow \text{getSecondClass}(A)\) \{get part\}
3: \(wp \leftarrow \text{getObjProp}(A)\) \{get wp type\}
4: \(NC_w \leftarrow \text{getNC}(w)\) \{obtain noun class whole\}
5: \(NC_p \leftarrow \text{getNC}(p)\) \{obtain noun class part\}
6: \(w_{pl} \leftarrow \text{pluralise}(w, NC_w)\) \{generate plural, using the pluraliser algorithm\}
7: \(NC'_w \leftarrow \text{getPlNC}(NC_w)\) \{obtain plural NC, from known list\}
8: \(a_w \leftarrow \text{getQCAll}(NC'_w)\) \{obtain quantitative concord (QC(all))\}
9: \(s_w \leftarrow \text{getSC}(NC'_w)\) \{obtain subject concord\}
10: \(\text{conjp} \leftarrow \text{phonoCondition}(\text{na}, p)\) \{prefix P with the CONJ, phonologically conditioned\}
11: \(r_p \leftarrow \text{getRC}(NC_p)\) \{obtain relative conc. for \(p\)\}
12: \(q_p \leftarrow \text{getQC}(NC_p)\) \{obtain quant. concord for \(p\) from the QC (exists)-list\}

Algorithm 7 Determine the verbalisation of basic whole-part in an axiom. Specifically: \(wp\) for spatial portions, without -dwa. (\(wp_{\text{spatial}}\))

1: input: two named classes that have the role \(w\) and \(p\), respectively
2: \(\text{commonFunctWP}(w, p)\)
3: if \(wp\) == spatial portion then
4: \(\text{RESULT} \leftarrow \text{'}a_w w_{pl} s_w conjp.'\) \{verbalise the axiom\}
5: end if
6: return RESULT

Algorithm 8 Determine the verbalisation of basic whole-part in an axiom. Specifically: participation with collectives, and \(w\) in singular (\(wp_{\text{cp}}\))

1: input: two named classes that have the role \(w\) and \(p\), respectively
2: \(\text{commonFunctWP}(w, p)\)
3: if \(wp\) == collective participation then
4: \(a_w \leftarrow \text{getQCAll}(NC_w)\) \{obtain quantitative concord (QC(all))\}
5: \(s_w \leftarrow \text{getSC}(NC_w)\) \{obtain subject concord\}
6: \(\text{RESULT} \leftarrow \text{'}a_w w s_w conjp r_p q_p dwa.'\) \{verbalise the axiom\}
7: end if
8: return RESULT

Algorithm 9 Determine the verbalisation of basic whole-part in an axiom. That is: subquantities [as parts] in singular, and no -dwa (\(wp_{\text{s}}\))

1: input: two named classes that have the role \(w\) and \(p\), respectively
2: \(\text{commonFunctWP}(w, p)\)
3: if \(wp\) == subquantities then
4: \(a_w \leftarrow \text{getQCAll}(NC_w)\) \{obtain quantitative concord (QC(all))\}
5: \(s_w \leftarrow \text{getSC}(NC_w)\) \{obtain subject concord\}
6: \(\text{RESULT} \leftarrow \text{'}a_w w s_w conjp.'\) \{verbalise the simple axiom\}
7: end if
8: return RESULT
Algorithm 10 Determine the verbalisation of basic whole-part in an axiom. Specifically: solid portion has \( W \) in singular, and the \( P \) with the PC, assuming that the part-quantity component is one word only (\( wp_{\text{solid}_p} \)).

**Require:** axiom of the form \( W \sqsubseteq \exists wp.P \) has been retrieved for verbalisation

1. \( w \leftarrow \text{getFirstClass}(A) \) \{ get whole \}
2. \( p \leftarrow \text{getSecondClass}(A) \) \{ get part \}
3. \( wp \leftarrow \text{getObjProp}(A) \) \{ get \( wp \) type \}
4. \( \text{if } wp == \text{solid portion} \text{ then} \)
5. \( NC_w \leftarrow \text{getNC}(w) \) \{ obtain noun class whole \}
6. \( q \leftarrow \text{first word of } p \) \{ \( p \) is typically a noun phrase or compound noun, first part the quantity, like slice, sample, etc \}
7. \( \text{if } \text{length}(p) == 2 \text{ then} \)
8. \( \text{stuff} \leftarrow \text{second word of } p \)
9. \( \text{else} \)
10. \( \text{stuff} \leftarrow \text{remainder of } p \)
11. \( \text{end if} \)
12. \( NC_q \leftarrow \text{getNC}(q) \) \{ obtain noun class quantity \}
13. \( a_w \leftarrow \text{getQCAll}(NC_w) \) \{ obtain quantitative concord (QC(all)) \}
14. \( s_w \leftarrow \text{getSC}(NC_q) \) \{ obtain subject concord \}
15. \( \text{conjp} \leftarrow \text{phonoCondition(‘na’, } q \text{)} \) \{ prefix the quantity-part of \( P \) with the CONJ, phonologically conditioned \}
16. \( \text{pc}_q \leftarrow \text{getPC}(q) \) \{ obtain possessive conc. for \( q \), for the ‘of’ \}
17. \( \text{os} \leftarrow \text{phonoCondition(pc}_q, \text{stuff}) \) \{ generate “of stuff” \}
18. \( r_p \leftarrow \text{getRC}(NC_q) \) \{ obtain relative conc. for \( p \) \}
19. \( \text{Result} \leftarrow ‘ a_w w s_w \text{conjp os } r_p q_p \text{dwa.’} \) \{ verbalise the axiom \}
20. \( \text{end if} \)
21. \( \text{return Result} \)

Algorithm 11 Determine the verbalisation of basic whole-part in an axiom. Specifically: constitution, of the built type (renamed this function after the inlg16). \( (\text{const}_a) \)

**Require:** axiom of the form \( W \sqsubseteq \exists wp.P \) has been retrieved for verbalisation

1. \( w \leftarrow \text{getFirstClass}(A) \) \{ get whole \}
2. \( p \leftarrow \text{getSecondClass}(A) \) \{ get part \}
3. \( wp \leftarrow \text{getObjProp}(A) \) \{ get \( wp \) type \}
4. \( \text{if } wp == \text{built constitution} \text{ then} \)
5. \( NC_w \leftarrow \text{getNC}(w) \) \{ obtain noun class whole \}
6. \( w_{pl} \leftarrow \text{pluralise}(w, NC_w) \) \{ generate plural, using the pluraliser algorithm \}
7. \( NC'_w \leftarrow \text{getPNC}(NC_w) \) \{ obtain plural NC, from known list \}
8. \( a_w \leftarrow \text{getQCAll}(NC'_w) \) \{ obtain quantitative concord (QC(all)) \}
9. \( sv \leftarrow \text{phonoCondVerb(‘akhiwe’, } NC'_w \text{)} \) \{ add SC + phono. cond. for vowel-commencing verbs \}
10. \( op \leftarrow \text{phonoCondition(‘nga’, } p \text{)} \) \{ generate “of part” \}
11. \( \text{Result} \leftarrow ‘ a_w w_{pl} sv \text{op } .‘ \) \{ verbalise the axiom \}
12. \( \text{end if} \)
13. \( \text{return Result} \)
Algorithm 12 Determine the verbalisation of basic whole-part in an axiom. Specifically: constitution as well, for other 'non-construction' constitution. (const_e)

Require: axiom of the form $W \sqsubseteq \exists wp. P$ has been retrieved for verbalisation

1: $w \leftarrow \text{getFirstClass}(A)$ \hspace{1cm} \{get whole\}
2: $p \leftarrow \text{getSecondClass}(A)$ \hspace{1cm} \{get part\}
3: $wp \leftarrow \text{getObjProp}(A)$ \hspace{1cm} \{get wp type\}
4: if $wp ==$ the other constitution then
5: \hspace{1cm} $NC_w \leftarrow \text{getNC}(w)$ \hspace{1cm} \{obtain noun class whole\}
6: \hspace{1cm} $wp_{pl} \leftarrow \text{pluralise}(w, NC_w)$ \hspace{1cm} \{generate plural, using the pluraliser algorithm\}
7: \hspace{1cm} $NC'_w \leftarrow \text{getPlNC}(NC_w)$ \hspace{1cm} \{obtain plural NC, from known list\}
8: \hspace{1cm} $a_w \leftarrow \text{getQCAll}(NC'_w)$ \hspace{1cm} \{obtain quantitative concord (Qc(all))\}
9: \hspace{1cm} $sv \leftarrow \text{phonoCondVerb('enziwe', NC'_w)}$ \hspace{1cm} \{add SC + phono. cond. for vowel-commencing verbs\}
10: \hspace{1cm} $op \leftarrow \text{phonoCondition('nga', p)}$ \hspace{1cm} \{generate "of part"\}
11: \hspace{1cm} $\text{RESULT } \leftarrow a_w \ wp_{pl} \ sv \ op$ \hspace{1cm} \{verbalise the axiom\}
12: end if
13: return $\text{RESULT}$
3.2 Part-whole reading direction

For the sake of presentation, also here we put the common functions in a separate algorithm that is used by the others (Algorithm 13).

**Algorithm 13** Common functions for pw verbalisation, commonFunctPW.

**Require:** $\mathcal{C}$ set of classes, language $\mathcal{L}$, $\sqsubseteq$ for subsumption, $\exists$ for existential quantification; variables: $A$ axiom, $NC_i$ noun class, $w, p \in \mathcal{C}$, $o \in \mathcal{R}$, $a_w$ a term; $r_p, q_p$ concords;

**Require:** axiom of the form $P \subseteq \exists pw.W$ has been retrieved for verbalisation

1: $p \leftarrow \text{getFirstClass}(A)$ \{get whole\}
2: $w \leftarrow \text{getSecondClass}(A)$ \{get part\}
3: $pw \leftarrow \text{getObjProp}(A)$ \{get pw type\}
4: $NC_p \leftarrow \text{getNC}(p)$ \{obtain noun class whole\}
5: $NC_w \leftarrow \text{getNC}(w)$ \{obtain noun class part\}
6: $p_pl \leftarrow \text{pluralise}(p, NC_p)$ \{generate plural, using the pluraliser algorithm\}
7: $NC'_p \leftarrow \text{getNC}(NC_p)$ \{obtain plural NC, from known list\}
8: $a_p \leftarrow \text{getQCAll}(NC'_p)$ \{obtain quantitative concord (QC(all))\}
9: $s_p \leftarrow \text{getSC}(NC'_p)$ \{obtain subject concord\}
10: $r_w \leftarrow \text{getRC}(NC_w)$ \{obtain relative conc. for w\}
11: $q_w \leftarrow \text{getQC}(NC_w)$ \{obtain quant. concord for w from the QC (exists)-list\}

**Algorithm 14** Determine the verbalisation of basic part-whole in an axiom. Specifically: structural, involvement, membership, part-subquantities, participation, part-whole relations.

1: **input:** two named classes that have the role $w$ and $p$, respectively
2: commonFunctPW($p, w$)
3: if $pw ==$ generic part then
4: \hspace{1cm} $pc = \text{\textquoteleft ya\textquoteright}$ \{no look-up needed for the PC, because it’s always ya- because always ingxenye (nc9)\}
5: \hspace{1cm} $pcw \leftarrow \text{phonoCondition}(pc, w)$
6: \hspace{1cm} \text{RESULT} $\leftarrow$ \textquoteleft $a_p$ $p_pl$ $s_p$ ingxenye $pcw$ $r_wq_w$ dwa. \textquoteright \{verbalise the axiom\}
7: end if
8: return \text{RESULT}
Algorithm 15 Determine the verbalisation of basic part-whole in an axiom. Specifically: part-whole, in the singular as well, to cater for subquantities that can be both mass and count noun, depending on context. (pw_s)

1: **input:** two named classes that have the role \( w \) and \( p \), respectively
2: \( \text{commonFunctPW}(p, w) \)
3: if \( pw == \) subquantity of then
4: \( a_p \leftarrow \text{getQCAll}(NC_p) \) \{ obtain quantitative concord (QC(all)) \}
5: \( s_p \leftarrow \text{getSC}(NC_p) \) \{ obtain subject concord \}
6: \( pc = 'ya' \) \{ no look-up needed for the PC, because it's always ya- because always ingxenye (nc9) \}
7: \( \text{RESULT} \leftarrow 'a_p \ s_p \ yingxenye \ pcw.' \) \{ verbalise the axiom \}
8: end if
9: **return** RESULT

Algorithm 16 Determine the verbalisation of basic part-whole in an axiom. Specifically: solid portion-of. (pw_solid_p)

**Require:** axiom of the form \( P \subseteq \exists \text{pw}.W \) has been retrieved for verbalisation

1: \( p \leftarrow \text{getFirstClass}(A) \) \{ get whole \}
2: \( w \leftarrow \text{getSecondClass}(A) \) \{ get part \}
3: \( \text{pw} \leftarrow \text{getObjProp}(A) \) \{ get pw type \}
4: if \( \text{pw} == \) solid portion then
5: \( q \leftarrow \text{first word of } p \) \{ \( p \) is typically a noun phrase or compound noun, first part the quantity, like slice, sample, etc \}
6: if \( \text{length}(p) == 2 \) then
7: \( \text{stuff} \leftarrow \text{second word of } p \)
8: else
9: \( \text{stuff} \leftarrow \text{remainder of } p \)
10: end if
11: \( NC_q \leftarrow \text{getNC}(q) \) \{ obtain noun class quantity \}
12: \( NC_w \leftarrow \text{getNC}(w) \) \{ obtain noun class whole \}
13: \( NC'_q \leftarrow \text{getPINC}(NC_q) \)
14: \( a_q \leftarrow \text{getQCAll}(NC'_q) \) \{ obtain quantitative concord (QC(all)) \}
15: \( \text{pc}_q \leftarrow \text{getPC}(q) \)
16: \( os \leftarrow \text{phonoCondition}(\text{pc}_q, \text{stuff}) \)
17: \( qpl \leftarrow \text{pluralise}(q, NC'_q) \)
18: \( s_q \leftarrow \text{getSC}(NC'_q) \)
19: \( pc = 'sa' \) \{ no look-up needed for the PC, because it's always sa- because always isiqephu (nc7) \}
20: \( \text{pcw} \leftarrow \text{phonoCondition}(pc,w) \)
21: \( r_w \leftarrow \text{getRC}(NC_w) \) \{ obtain relative conc. for \( w \) \}
22: \( q_w \leftarrow \text{getQC}(NC_w) \) \{ obtain quant. concord for \( w \) from the QC (exists)-list \}
23: \( \text{RESULT} \leftarrow 'a_q \ qpl \ os \ s_q \ isiqephu \ pcw \ r_wq_w \ dwa.' \) \{ verbalise the axiom \}
24: end if
25: **return** RESULT
Algorithm 17 Determine the verbalisation of basic part-whole in an axiom. Specifically: spatial portion-of. \((\text{pw\_spatial\_p})\)

1: input: two named classes that have the role \(w\) and \(p\), respectively
2: \(\text{commonFunctPW}(p, w)\)
3: if \(pw == \) spatial portion of then
4: \(pc = 'wa'\) \{no look-up needed for the PC, because it's always wa- because always umunxa (nc3)\}
5: \(pcw \leftarrow \text{phonoCondition}(pc, w)\)
6: \(\text{RESULT} \leftarrow 'a_p p_pl s_p\text{ngununxa }pcw.'\) \{verbalise the axiom\}
7: end if
8: return RESULT

Algorithm 18 Determine the verbalisation of basic part-whole in an axiom. Specifically: participates-in, for collective parts, in singular. \((\text{pw\_pi\_c})\)

1: input: two named classes that have the role \(w\) and \(p\), respectively
2: \(\text{commonFunctPW}(p, w)\)
3: if \(pw == \) collective participates in then
4: \(a_p \leftarrow \text{getQCAll}(NC_p)\) \{obtain quantitative concord (QC(all))\}
5: \(s_p \leftarrow \text{getSC}(NC_p)\) \{obtain subject concord\}
6: \(lpre \leftarrow \text{phonoCondLocPrefix}(w, NC_w)\) \{add locative prefix to whole (if nc = 1a, 2a, 3a, or 17 then ku+word, else e+word)\}
7: \(lpreWlsuf \leftarrow \text{phonoCondLocSuffix}(lpre)\) \{add locative suffix to whole (the -ini/-eni/-wini etc)\}
8: \(\text{RESULT} \leftarrow 'a_p p slhlanganyele \text{lpreWlsuf }r_wq_wdwa.'\) \{verbalise the axiom\}
9: end if
10: return RESULT

Algorithm 19 Determine the verbalisation of basic part-whole in an axiom. Specifically: contained-in. \((\text{pw\_ci})\)

1: input: two named classes that have the role \(w\) and \(p\), respectively
2: \(\text{commonFunctPW}(p, w)\)
3: if \(pw == \) contained in then
4: \(Wlsuf \leftarrow \text{phonoCondLocSuffix}(w)\) \{add locative suffix to whole (the -ini/-eni/-wini etc)\}
5: \(lpreWlsuf \leftarrow \text{phonoCondLocPrefix}(Wlsuf, NC_w)\) \{add locative prefix to whole (if nc = 1a, 2a, 3a, or 17 then ku+word, else e+word)\}
6: \(\text{RESULT} \leftarrow 'a_p p pl\text{lpreWlsuf }r_wq_wdwa.'\) \{verbalise the axiom\}
7: end if
8: return RESULT
4 Phonological conditioning

This section first lists the phonological conditioning rules that have been implemented at the
time of writing and which have been mentioned informally in, mainly [5] for locatives and the
vowel-commencing verb roots, but not the others. They still seem incomplete and therefore also
listed outside their algorithm environment. The algorithms are presented afterwards.

- vowel coalescence function (\textit{phonoCondition} in the algorithms), where \(X\) and \(Y\) are the
  remainder of the word:
  \begin{itemize}
    \item \(Xa + aY \rightarrow XaY\)
    \item \(Xa + (iY \mid eY) \rightarrow XeY\)
    \item \((Xa, X \neq ng) + uY \rightarrow XoY\) // the ‘\(X \neq ng\)’ is an old remnant. as there’s also nga
      + uY = ngoY now, so can be deleted. (or not?)
    \item \(Xe + aY \rightarrow XaY\)
    \item \(Xe + iY \rightarrow XeY\)
    \item \(Xe + (oY \mid uY) \rightarrow XoY\)
    \item \(nga + oY \rightarrow ngoY\) //can this be generalised, as -a + o- = -o-?
    \item \(nga + uY \rightarrow ngoY\) //can be included in the 3rd one
  \end{itemize}

- locative prefix (\textit{phonoCondLocPrefix} in the algorithms), possibly still incomplete
  \begin{itemize}
    \item if nc = 1a, 2a, 3a, or 17 \(\rightarrow\) ku+\textit{word} (subject to the vowel coalescence listed in the
      previous item)
    \item for other ncs \(\rightarrow\) e+\textit{word} (subject to the vowel coalescence listed in the previous item)
  \end{itemize}

- locative suffix (\textit{phonoCondLocSuffix} in the algorithms)
  \begin{itemize}
    \item regular cases:
      \begin{itemize}
        \item \(* Xa \rightarrow Xeni\)
        \item \(* Xe \rightarrow Xeni\)
        \item \(* Xi \rightarrow Xini\)
        \item \(* Xo \rightarrow Xweni\)
        \item \(* Xu + (\sim ph) \rightarrow Xwini\) // the ph in second and third last position
        \item \(* Xu + ph \rightarrow Xshini\) // the ph in second and third last position
        \item \(* otherwise\ \textit{word}+\textit{ini}\)
      \end{itemize}
    \item exceptions: imvilophu, idiphu, ifomu \(\rightarrow\) \textit{word}[0:-1] + ini.
  \end{itemize}

- subject concord (\textit{sc}) and vowel-commencing verb stem (\textit{word}) and stem-minus-first-letter
  (\(X\)), named \textit{phonoCondVerb} in the algorithms in the preceding sections:
  \begin{itemize}
    \item \((\text{length}(\textit{sc}) \geq 2, \sim ku,lu) + (aX \mid eX) \rightarrow [b/l/s/z]\textit{word}\) // the remaining sc
      consonants (\textit{sc}[:-1] to be more precise)
    \item \(a + (aX \mid eX) \rightarrow \textit{word}\)
    \item \(i + (aX \mid eX) \rightarrow y\textit{word}\)
    \item \(u + (aX \mid eX) \rightarrow w\textit{word}\)
    \item \(ku,lu + (aX \mid eX) \rightarrow [k/l]\textit{wword}\)
    \item \((\text{length}(\textit{sc}) \geq 2, \sim ku) + oX \rightarrow [b/l/s/z]\textit{word}\) // the remaining sc consonants
      (\textit{sc}[:-1] to be more precise)
  \end{itemize}
\(- a,i + oX \rightarrow \text{word} \)
\(- u + oX \rightarrow \text{wword} \)
\(- ku + oX \rightarrow \text{kword} \)

- negative subject concord (negsc) and vowel-commencing verb stem (word), in the algorithm as phonoCondNegSc:
  
\(- (aX \mid eX) \rightarrow \text{negscyword} \)
\(- (iX \mid oX \mid uX) \rightarrow \text{negsengword} \)

Also in this case, the algorithms were gradually extended in the code, so there may be some duplication (see also above) that may have yet to be refactored.

**Algorithm 20 (VowelCoal)**  Vowel coalescence (or: two [the last letter of the first part and the first letter of the second part] becoming one)

1: **input:** two strings, first and second, respectively, where the former is to be agglutinated to the latter into a new word.
2: if \( f_{[-1]} == 'a' \) and \( s_0 == 'a' \) then
3: \( n \leftarrow f_{[0:-1]} \text{as}[1] \) \{ \( a+a = a \) \}
4: else if \( f_{[-1]} == 'a' \) and \( (s_0 == 'i' \text{ or } s_0 == 'e') \) then
5: \( n \leftarrow f_{[0:-1]} \text{es}[1] \) \{ \( a+i/e = e \) \}
6: else if \( f_{[-1]} == 'a' \) and \( f \neq 'nga' \) and \( s_0 == 'u' \) then
7: \( n \leftarrow f_{[0:-1]} \text{os}[1] \) \{ \( a+u = o \) \}
8: else if \( f_{[-1]} == 'e' \) and \( s_0 == 'a' \) then
9: \( n \leftarrow f_{[0:-1]} \text{as}[1] \) \{ \( e+a = a \) \}
10: else if \( f_{[-1]} == 'e' \) and \( s_0 == 'i' \) then
11: \( n \leftarrow f_{[0:-1]} \text{es}[1] \) \{ \( e+i = e \) \}
12: else if \( f_{[-1]} == 'e' \) and \( (s_0 == 'o' \text{ or } s_0 == 'u') \) then
13: \( n \leftarrow f_{[0:-1]} \text{os}[1] \) \{ \( e+o/u = o \) \}
14: else if \( f_{[-1]} == 'u' \) then
15: \( n \leftarrow f_{s}[1] \) \{ assuming the u is a ‘stronger’ vowel, for now \}
16: else
17: if \( f == 'nga' \) and \( s_0 == 'o' \) then
18: \( n \leftarrow \text{ngos}[1] \)
19: else if \( f == 'nga' \) and \( s_0 == 'u' \) then
20: \( n \leftarrow \text{ngos}[1] \)
21: else
22: \( n \leftarrow \text{other} \) \{ sentinel word to detect a phonological conditioning not covered yet \}
23: end if
24: end if
25: return \( n \)
Algorithm 21 (LocPre) Locative prefix for the noun or named entity.

1: \textbf{input:} word $w$ and noun class $nc$
2: \hspace{0.5cm} $l \leftarrow ew$ \hspace{1cm} \{default case\}
3: \hspace{0.5cm} \textbf{if $nc == 1a$ or $nc == 2a$ or $nc == 3a$ or $nc == 17$ then}
4: \hspace{1cm} $l \leftarrow \text{VowelCoal}('ku', w)$
5: \hspace{0.5cm} \textbf{else}
6: \hspace{1cm} $l \leftarrow \text{VowelCoal}('e', w)$
7: \hspace{0.5cm} \textbf{end if}
8: \hspace{0.5cm} \textbf{return $l$}

Algorithm 22 (LocSuf) Locative suffix for the noun or named entity.

1: \textbf{input:} word $w$
2: \hspace{0.5cm} \textit{exceptions} = ['imvilophu', 'idiphu', 'ifomu']
3: \hspace{0.5cm} \textbf{if $w \in \text{exceptions}$ or $w[-1] == i$ then}
4: \hspace{1cm} $l \leftarrow w[0:-1]ini$ \hspace{1cm} \{note: 'ini' is the common case\}
5: \hspace{0.5cm} \textbf{else if $w[-1] == a$ or $w[-1] == e$ then}
6: \hspace{1cm} $l \leftarrow w[0:-1]eni$ \hspace{1cm} \{-a/-e = eni\}
7: \hspace{0.5cm} \textbf{else if $w[-1] == o$ then}
8: \hspace{1cm} $l \leftarrow w[0:-1]weni$ \hspace{1cm} \{-o = weni\}
9: \hspace{0.5cm} \textbf{else if $w[-1] == u$ and $w[-3:-2] != ph$ then}
10: \hspace{1cm} $l \leftarrow w[0:-1]wini$ \hspace{1cm} \{-u = wini\}
11: \hspace{0.5cm} \textbf{else if $w[-1] == u$ and $w[-3:-2] == ph$ then}
12: \hspace{1cm} $l \leftarrow w[0:-3]shini$ \hspace{1cm} \{-u = shini\}
13: \hspace{0.5cm} \textbf{else}
14: \hspace{1cm} $l \leftarrow wini$
15: \hspace{0.5cm} \textbf{end if}
16: \hspace{0.5cm} \textbf{return $l$}
Algorithm 23 (VowelVerb) Phonological conditioning for conjugation (SC with a vowel-commencing verb root).

1: input: word w and its noun class nc
2: sc ← getSC(nc) \{get subject concord for that NC\}
3: if \((w[0] == 'a' \lor w[0] == 'e') \land \text{length}(sc) \geq 2 \land sc != 'ku' \land sc != 'lu'\) then
4: conjv ← sc[-1]w \{long sc + a-/e- = drop last letter of sc\}
5: else if \((w[0] == 'a' \lor w[0] == 'e')\) and \(sc == 'a'\) then
6: conjv ← w \{a+a-/e- = drop sc\}
7: else if \((w[0] == 'a' \lor w[0] == 'e')\) and \(sc == 'i'\) then
8: conjv ← yw \{i+a-/e- = y+a-/e-\}
9: else if \((w[0] == 'a' \lor w[0] == 'e')\) and \(sc == 'u'\) then
10: conjv ← wu \{u+a-/e- = w+a-/e-\}
11: else if \((w[0] == 'a' \lor w[0] == 'e')\) and \((sc == 'ku' \lor sc == 'lu')\) then
12: conjv ← sc[0]ww \{ku/lu+a-/e- = k/l+w+a-/e-\}
13: else if \(w[0] == 'o'\) and \(\text{length}(sc) \geq 2\) and \(sc != 'ku'\) then
14: conjv ← sc[-1]w \{long sc + o- = drop last letter of sc\}
15: else if \(w[0] == 'o'\) and \((sc == 'a' \lor sc == 'i')\) then
16: conjv ← w \{i+a-o- = o-\}
17: else if \(w[0] == 'o'\) and \(sc == 'u'\) then
18: conjv ← wu \{u + o- = w + o-\}
19: else if \(w[0] == 'o'\) and \(sc == 'ku'\) then
20: conjv ← kw \{ku + o- = kw + o-\}
21: else
22: conjv ← w \{or: don’t do anything\}
23: end if
24: return conjv

Algorithm 24 (NegVowelVerb) Phonological conditioning for negated conjugation (NEG SC with a vowel-commencing verb root).

1: input: word w and its noun class nc
2: nsc ← getNEGSC(nc) \{get negative subject concord for that NC\}
3: if \(w[0] == 'a' \lor w[0] == 'e'\) then
4: negconjv ← nscyw \{anything + a-/e- = anything + y + a-/e-\}
5: else
6: negconjv ← nscngw \{anything + i-/o-/u- = anything + ng + i-/o-/u-\}
7: end if
8: return negconjv
5 Architecture of the verbaliser

The OWL verbaliser is described in [7]. The architecture is depicted in Figure 2 and is such that one can:

- Run the Python code in the interpreter, feeding it just the strings in the format given by the definitions in the code;
- Use Owlready to fetch the vocabulary from an ontology stored in OWL/XML format, where the output is written to stdout/console;
- Use Owlready and Tkinter to fetch the vocabulary from an ontology stored in OWL/XML format and get pretty printing in colour.

To use the software, see the readme.txt in the zipfile for instructions.

![Figure 2: Principal components of the OWL verbaliser. (Source: [7])](image)

Acknowledgements

As can be seen for the references, the main collaborator in trying to structure the linguistic knowledge is Langa Khumalo, and I hereby thank him for the collaboration.

This work is based on the research supported in part by the National Research Foundation of South Africa (CMK: Grant Number 93397).
References


