Handling Reduplication in a Morphological Analyzer for Wamesa

Emily Lin

December 20, 2019
Version: Final Defense Draft
Advisor: Jane Chandlee

Abstract

This thesis seeks to assess previous computational work done regarding reduplication in order to account for reduplication in a morphological analyzer for the Wamesa language. This paper includes a brief introduction to Wamesa, reduplication, two-level rules, and morphological analysis. The use of finite-state transducers as morphological analyzers is also discussed. Additionally, various methods that researchers have used to deal with reduplication in computational models are evaluated. Methods include the use of two-level rules and supplementing finite-state transducers with an equality operator, memory devices, and bidirectional reading capabilities. Ultimately, I propose additions and modifications to the Wamesa morphological analyzer, which involves the use of the Helsinki Finite-State Toolkit and two-level rules.

Acknowledgements

I would like to thank Jane Chandlee for being an incredible thesis advisor; Emily Gasser for inspiring me to work with Wamesa and giving me access to the plethora of knowledge and data that she has created and compiled; Jonathan Washington for giving me insight into his use of finite-state transducers for morphological analysis and giving suggestions for my thesis work; and Daniel Swanson for building a morphological analyzer for Wamesa and giving me permission to use it. I would also like to thank Brook Lillehaugen, Sorelle Friedler, and the other incredible faculty in the computer science and linguistics departments that have been incredibly helpful and supportive along the way.

Contents

1 Introduction

Wamesa is a highly under-resourced and under-documented indigenous language spoken in Indonesia. Currently, Professor Emily Gasser is the primary linguist studying and documenting Wamesa as well as producing materials for linguistic study (Gasser, 2014). That said, few technological resources exist for it. Due to the political dominance of other languages located in the vicinity, fewer and fewer children have an active command of the language or know it at all (Gasser, 2014). Thus, the number of speakers of Wamesa is decreasing, and the language is considered endangered.

Given that Wamesa is an endangered and under-documented language, morphological analysis[1] of Wamesa is particularly important. Along those lines, automating the process of morphological analysis using a computational model is helpful, since the model would allow for faster and more comprehensive understanding of the language. In addition to being able to generate and recognize words, the morphological analyzer, the computational model built to handle morphological analysis, is essential to a range of applications. For example, the analyzer can be used for applications such as spellchecking, translation, speech-to-text, and text-to-speech. Thus, they are a powerful technological tool that can augment the documentation and understanding of languages, which are key to their preservation.

In Spring 2019, Daniel Swanson, one of Gasser's students, began developing a morphological analyzer for Wamesa. In this thesis, I examine how the analyzer can be modified to account for the process of reduplication. While there are few instances of reduplication in Wamesa, reduplication is very common in Austronesian languages (Gasser, 2014). Historically, accounting for reduplication in morphological analyzers has proven to be a computationally challenging task, in contrast with other linguistic phenomena. That said, hopefully the research done in thesis can help inform how reduplication can be computationally handled for other languages, particularly those which are closely related to Wamesa.

In the following sections, I will provide more information about the Wamesa language and reduplication. I will also introduce morphological analysis, two-level morphology, and finite state transducers (FSTs). In addition, I discuss the various

[1]Morphological analysis is the process of determining the structure, components, and meaning of words.

tools and methods that researchers have used to handle reduplication in computational models. Ultimately, I propose two-level rules that can be added to the existing morphological analyzer for Wamesa in order to account for reduplication.

1.1 Wamesa

Wamesa is an Austronesian language that is part of the South Halmahera-West New Guinea (SHWNG) subgroup (Gasser, 2014). It has 5,000-8,000 speakers and is spoken in West Papua, a province of Indonesia. Wamesa's orthography (writing system) is based primarily on that of Indonesian, and the three major dialects of Wamesa are Windesi, Wandamen, and Bintuni. This paper describes the Windesi dialect, except where noted otherwise. In the following sections, I will give a brief overview of the sound inventory and morphology of Wamesa.

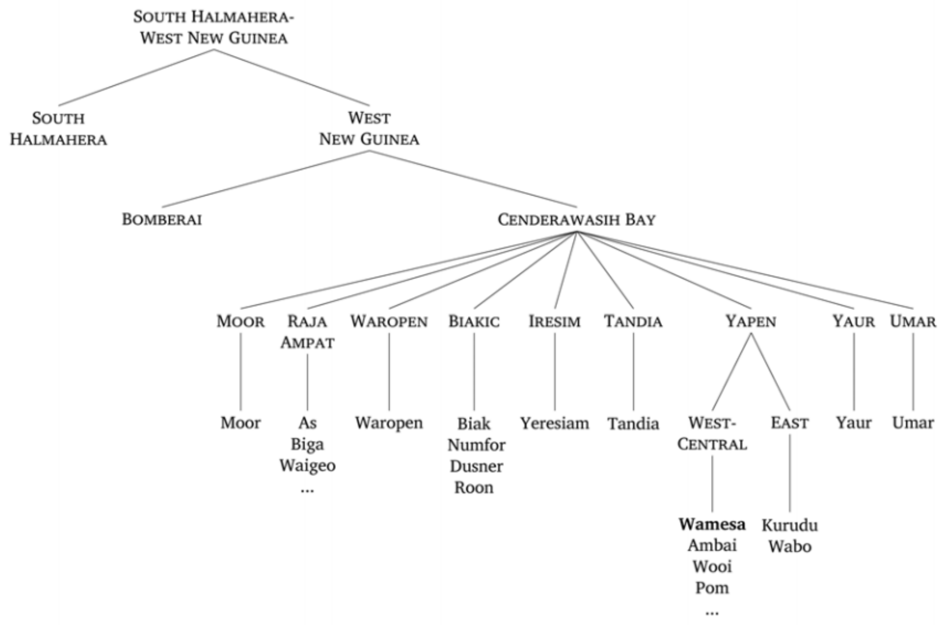


Fig. 1.1: Tree of South Halmahera-Western New Guinea Languages (Gasser, 2014)

1.1.1 Sound Inventory

Wamesa has five contrastive vowels (i, e, a, u, o) and 14 consonants (Gasser, 2014). The diphthongs are /au/, /ai/, /ei/, /oi/, /ui/, /ou/, and /uo/. The IPA[2] charts for Wamesa's sound inventory appear below.

[2]IPA stands for International Phonetic Alphabet. It consists of symbols designed to represent all sounds in human languages.

	Bilabial	Alveo-dental	Velar
Nasal	m	n	ŋ
Plosive	p b	t d	k (g)
Fricative	β	s	
Affricate		(dʒ)	
Tap/Trill		r	
Lateral		(l)	

Fig. 1.2: Consonant Inventory (Gasser, 2014)

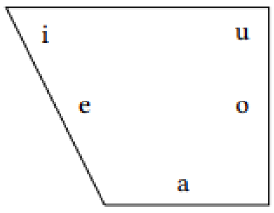


Fig. 1.3: Vowel Inventory (Gasser, 2014)

As for stress, Wamesa is a bounded language (stress alternates between syllables), and the location of primary stress is not predictable (Gasser, 2014). This means that stress must be specified in the underlying form of words. However, stress shift can occur to create better phonological structures, words which sound better due to rhythm or stress alternation.

1.1.2 Morphology

Wamesa is an SVO (subject verb object) language (Gasser, 2014). It does not have case or specificity. It requires subject agreement, which distinguishes between person (1st, 2nd, 3rd), number (singular, dual, or plural), human or nonhuman, and inclusive or exclusive. Affixes include the applicative, essive[3], and causative. Clitics include definite determiners, locatives, a topicalizer, and negation. Morphophonological rules in Wamesa include mid-vowel raising (in the underlying sequence /eu/, the mid-vowel /e/ becomes the high vowel [i] in the surface form, resulting in /eu/ -> [iu]) and cluster simplification (when two consonants are adjacent in the underlying form, the affix-final consonant is deleted).

[3]Essives indicate a state of being. In Wamesa, the essive *ve-* is prefixed to a word X to mean 'having the properties of X' (Gasser, 2014).

1.2 Reduplication

Reduplication is a morphological phenomenon that is defined as the copying of phonological constituents to create a word with a different meaning. Both total reduplication, the copying of a full word, and partial reduplication, when only part of a word is copied, commonly occur in the world's languages. Additionally, reduplication can be productive, which means that new words in the language can be generated via reduplication. In English, speakers use total reduplication, and the most common use of reduplication is for contrastive focus (Ghomeshi et al., 2004).

For example,

(1) a. I like him a lot.

b. Like? Or like-like?

For some native speakers of English, *like* entails platonic feelings while *like-like* entails romantic and/or sexual interest. Thus, *like* and *like-like* have different meanings, although the latter was generated from reduplicating the first.

1.2.1 Reduplication in Wamesa

In Wamesa, reduplication has limited productivity. In 2014, only 8 of nearly 1000 lexical entries that Gasser found were identified as being reduplicable (Gasser, 2014). Additionally, Wamesa has not been found to have total reduplication; it has partial reduplication which, in Wamesa, involves copying the first or second syllable of the word that is being reduplicated. Partial reduplication has been found in adjectives, verbs, and adverbs in Wamesa. In these cases, the reduplication works to intensify words.

Examples of reduplication in Wamesa appear below, where reduplicating (a) results in (b). The following examples are from Gasser's dissertation:

(2) a. *masabu* 'broken, cracked'

b. *ma**sa**sabu* 'smashed, shattered'

(3) a. *kasiou* 'angry'

b. *ka**si**siou* 'furious'

(4) a. *maraba* 'ripped; holey'

b. *ma**ra**raba* 'ripped to shreds; full of holes'

(5) a. *kavio* 'to talk'

b. *ka**vi**vio* 'to chatter on'

(6) a. *saira* 'quickly'

b. ***sa**saira* 'very quickly'

(7) a. *mase* 'hot'

b. ***ma**mase* 'very hot'

The following examples are from the Wamesa Talking Dictionary (Gasser, 2015) and consultation with Emily Gasser:

(8) a. *nganggau* 'dizzy, confused'

b. ***nga**nganggau* 'very dizzy, confused'

(9) a. *nganggat* 'to bother'

b. ***nga**nganggat* 'to bother a lot'

(10) a. *peya* 'to be quiet'

b. ***pe**peya* 'silent'

The following examples are from a dictionary of the Wandamen dialect (Henning et al., 1991). They are possibly instances of reduplication but have yet to be confirmed as such by speakers.

(11) a. *sianggono* 'red'

b. *si**ang**anggono* 'very red'

(12) a. *adiawa* 'to hear'

b. ***adi**diawa* 'to listen'

(13) a. *aka* 'to grope for in the dark'

b. *a**ka**ka* 'to grope in the dark'

(14) a. *panami* 'smooth'

b. *pa**ni**niami* 'very smooth'

(15) a. *kavo* 'speech, language, word'

b. *ka**va**vo* 'conversation, discussion'

1.3 Morphological Analysis

To understand morphological analysis, we first need to introduce morphology, the study of morphemes, which are the meaningful parts of words. For example, in the English language, the word *creations* consists of the morphemes *create*, *-tion*, and *-s*, all of which contribute to and are necessary for expressing the meaning of the word *creations*. In this case, the morpheme *-s* means plural, and the nominalizer *-tion* is suffixed to the verb root *create*, meaning *-tion* changes the verb *create* into a noun that can take on the following meanings: the act of creating as in (a) and the result of creating as in (b).

(16) a. The creation of art is crucial.

b. The creations are large and thought-provoking.

The goal of morphological analysis is to take words in a language and determine the morphemes they consist of as well as what the morphemes themselves mean. Going back to the word *creations*, we do not just want to know that it contains the morpheme *-s*, we also want to know that *-s* signifies pluralization. Additionally, note that *-s* is not always a morpheme nor does it always mean the same thing. For example, in the word *process*, *-s* is not a morpheme but rather just a character of the morpheme *process*. In the word *speaks*, the second <s> signifies 3rd person singular, as in *He speaks*. Thus, a morphological analyzer, a computational model which handles morphological analysis, for English would output `create + [nominalizer]`

`+ [plural]` when given the input *creations* to analyze. In the other direction, the model should generate and output *creations* when given the input `create + [nominalizer] + [plural]`.

Note that morphological analysis can be a computationally difficult task, as the orthographic representation of words may not contain the full orthographic representations of the morphemes from which they are composed. For example, we can observe that the word *creations* does not contain the full spelling of the morpheme *create*, which ends with an <e>. In other words, while the surface form[4] of a word is derived from the underlying or lexical form[5], it may not (and often does not) appear as a simple concatenation or stringing together of the underlying morphemes. In fact, not all morphology is concatenative and may instead involve processes such as deletion or substitution. For example, *rang* is the past tense of *ring*. Notice that letters are not being added. Instead, replacement occurs and results in a different word with a related but modified meaning. In summary, the morphology of words can vary widely, and morphological analysis is essential to understanding the meaning and structure of words and therefore language.

[4]The surface form refers to how a word actually sounds when spoken or how a word actually looks when written.

[5]The underlying or lexical form refers to the representation of words that includes grammatical markers such as person, number, case, and tense. For example, 'walk+past+1st person' is an underlying representation of the surface form *walked*.

Background 2

In this chapter, I introduce finite state transducers, which are used for morphological analysis, and computational challenges regarding reduplication. Additionally, I give an overview of two-level morphology and the Helsinki Finite State Toolkit (HFST), which serve as foundations for the Wamesa morphological analyzer.

2.1 Finite State Transducers (FSTs)

The automation of morphological analysis requires the development of computational models that can represent the many complexities of language. While finite state transducers (FSTs) can perform a wide range of tasks, they are especially popular for handling morphological analysis.

FSTs are mathematical models that are defined with an input alphabet, output alphabet, start state, set of initial states, set of final states, and a transition function. The transition function can be thought of as a set of rules specifying which set of states to go to given the current state and input. Deterministic FSTs can only have one transition arc per [state, input] pair while non-deterministic FSTs can contain states with multiple transitions arcs out of them for one given input. For example, if the machine is in a state A, upon reading an input x, a deterministic machine can only go to one of the states in the set $\{B, C, D\}$ while a non-deterministic machine may go to all three. As a distinction from the overarching class of finite state automata (FSA which also have states and transitions, FSTs write output in addition to reading input. At final states, reading and writing stops, and output is returned. A diagram of a simple FST appears below:

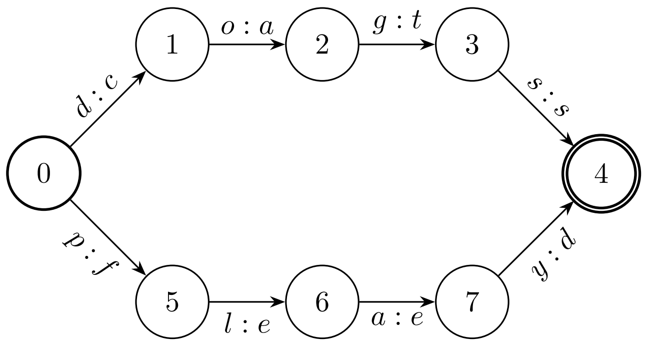


Fig. 2.1: A Simple FST (dogs->cats; play->feed)

Above, the start state is labeled 0, and the double circle of state four denotes it as the final or accept state. Each transition arrow, is labeled with a transformation. Letters on the left of the colon are part of the machine's input alphabet, and information to the right of the colon are part of the output alphabet. Note that these two alphabets can contain the same or completely different symbols. In this FST, an input string "dogs" would be transformed into "cats", and an input string "play" would be converted into "feed". This FST would only accept the inputs "dogs" and "play" since they are the only inputs that the machine has transitions for that end in an accept state. While the FST above only has one accept state, an FST may have multiple accept states. Additionally, there may be transitions that loop and remain in the same state.

The example I provided only transforms two words via one-to-one letter replacements, but a more sophisticated FST for morphological analysis would have many more alphabet characters, states, and transitions. This would allow the machine to account for more complicated transformations, such as generating the surface form of a word given its underlying representation containing grammatical tags. Regardless of the complexity or size of an FST, FSTs operate according to their defined transition functions. In fact, FSTs are powerful models for morphological analysis since they can handle both word generation (underlying to surface form) and analysis (surface to underlying form).

2.2 Computational Challenges Regarding Reduplication

Traditionally, the FSTs used for morphological analysis can only read input in the forward direction and are thus called 1-way FSTs (Dolatian and Heinz, 2018). In order to duplicate a word, the 1-way FST must memorize the input it has already seen. This requires that there be as many paths as there are strings (sequences of characters) that need to be copied (Roark et al., 2007). As long as the number of characters to memorize and copy is known or bounded, 1-way FSTs are able to generate and analyze words that have both partial and total reduplication.

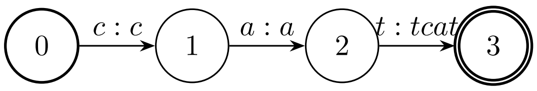


Fig. 2.2: A Simple FST (cat->catcat)

For example, if the above FST needed to generate the string *catcat* it must be given the input *cat*. Additionally, the FST would have to memorize each input character as it reads it and then output both what it read and memorized. Since the FST has a finite number of states, there must also be a finite number of inputs that it can recognize and generate. For example, the FST would not be able to generate the string *dogdog* unless we further expanded the FST as such:

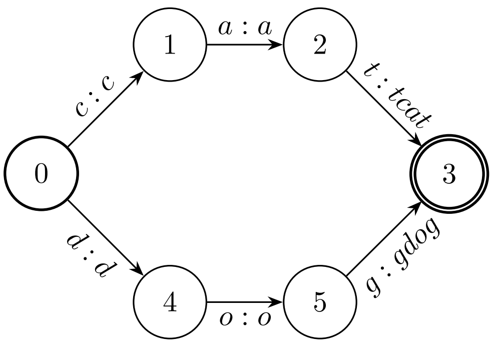


Fig. 2.3: A Simple FST (cat->catcat; dog->dogdog)

Notice that the expansion of the FST from Fig. 2.2 to Fig 2.3 still only accounts for two words. Handling additional words would require yet more expansion. The takeaway here is that no matter how much the FST is expanded, it will never be able to account for an infinite number of inputs.

That said, building a 1-way FST that handles total reduplication requires a significant increase in the number of states required, as the length of the words to be duplicated may vary and be unspecified. As a result, computational complexity, the amount of time and computer memory required for a given task, also increases when modeling total reduplication using a 1-way FST.

Additionally, reduplication can be productive, which means that new, previously unseen words can be generated. While not all reduplicated words may be considered grammatical or meaningful by speakers, productive reduplication is still an important aspect of many languages. Since 1-way FSTs must, by definition, have a finite number of states, they cannot model productive total reduplication, which has an infinite number of possibilities. In other words, 1-way FSTs are unable to use memorization to reduplicate words without knowing which and how many characters in a string need to be memorized (Roark et al., 2007).

Due to the limitations of traditional 1-way FSTs in handling total and productive reduplication, researchers have experimented with various ways of modifying FSTs. These efforts will be discussed later in Chapter 3, the Literature Review.

2.3 Two-level Rules

The understanding of two-level morphology is essential to discussing the computational task of morphological analysis. Two-level morphology was formalized by Kimmo Koskenniemi (1983) to allow computational models to handle morphological analysis. The term *two-level* refers to the two representations of a word (lexical and surface) which Koskenniemi deemed necessary for morphological analysis (Oflazer, 1994). The lexical level consists of stems and morphotactically permitted endings, including functional components that specify grammatical content (e.g. tense and agreement) (Koskenniemi, 1983). The surface level of a word is its written representation which specifies phonemes, the letters representing distinct sounds in a given language. The specification of both levels allows a computer to handle morphological phenomena such as inflection.

A morphological analyzer employing traditional rewrite rules instead of two-level rules would require an input to pass through multiple levels of transducers, each accounting for a rewrite rule (Karttunen and Beesley, 2001). In order to produce a correct output, a machine must apply these rewrite rules must be applied sequentially and in the correct order. In contrast, two-level rules can be applied by a machine simultaneously, which means that the ordering of two-level rules is not an issue and that computational time complexity is also decreased (Muhirwe and Trosterud, 2008). Additionally, two-level rules are bidirectional, which means that they can be used to both generate and analyze words (Koskenniemi, 1983). This is because word recognition involves finding a corresponding lexical representation, and word generation involves finding the corresponding surface representation (Koskenniemi, 1986). An example of a two-level rule from the Wamesa analyzer (Swanson, 2019) appears below:

1. "2SG infix 1"
   %{B%}:Cx <=> __ u: %>: Cx: ;
   where Cx in Consonants ;

This rule, which is involved in the infixation of the second person singular (2SG) marker, states that when appearing before an underlying /u/, the underlying %{B%} becomes a copy of the consonant following the /u/. Additional details regarding the meaning and notation of two-level rules will be discussed in the remainder of this section.

In 1986, Koskenniemi built a two-level morphological analyzer, termed the KIMMO model, based on his concept of two-level morphology and created a formalism for other researchers to follow. Koskenniemi (1986) details that the model must include a lexicon component specifying the morphophonological representations of words and suffixes in addition to a rule component containing two-level rules.

Koskenniemi (1986) states that the rule component of a two-level model must include the following:

1. A surface alphabet
2. Subsets of the surface alphabet to be used in the rules
3. A lexical alphabet
4. Subsets of the lexical alphabet
5. Definitions for abbreviations and subexpressions used in the rules
6. Two-level rules

Subsets are necessary because morphological rules often apply to a category or class of alphabet characters, rather than one individual character. Alphabet characters may be grouped into a subset of vowels, for example. Abbreviations and subexpressions allow researchers to have some flexibility in how they want to write two-level rules, although Koskenniemi's formalism provides certain standards to follow. Koskenniemi (1986) specifies in his formalism for two-level rules that the lexical level is written on the left of a semicolon, and the surface level is written on the right. For example, in **a:o**, **a** is the lexical form, and **o** is the surface form. When a lexical form corresponds to multiple surface forms, the forms are written within square brackets **[ ]**, and the 'or' symbol **|** is used to separate the forms. For example, **[:y | :i]**, specifies that the surface form may either be the phoneme /y/ or /i/. The class of vowels are indicated with **V**, and consonants are indicated with **C**. Additionally, the use of an asterisk * indicates zero or more instances of a character, and a plus sign + indicates a minimum of one instance. For example, **:C*** would be interpreted by the machine as a sequence of zero or more consonants in the surface form (Koskenniemi, 1986).

In addition to the two levels, a rule must also specify the environment in which a change is taking place. Rules must also be preceded by a name or description of what they do, e.g. "Vowel doubling."

Roark et al. (2007) provide a summary of two-level rules for the KIMMO system, stating that they are of the following form:

CorrespondencePair **operator** LeftContext __ RightContext

The correspondence pair is the lexical form followed by the surface form. The operator is a symbol specifying additional details of the rule. The left and right context specify what appears to the left and right of the location of the correspondence, indicated with an underscore _. The correspondence can be thought of as a morphological or phonological change converting a lexical form into a surface form. The operator is a symbol which indicates which of the following types of morphological rules is being written:

1. Exclusion rule a:b /<= LC __ RC
2. Context restriction rule a:b => LC __ RC
3. Surface coercion rule a:b <= LC __ RC
4. Composite rule a:b <=> LC __ RC

According to Roark et al. (2007), the interpretation of these rules is as follows:

1. Exclusion rule: *a* cannot be realized as *b* in the given context.
2. Context restriction rule: *a* can only be realized as *b* in the given context and nowhere else.
3. Surface coercion rule: *a* must be realized as *b* in the given context.
4. Composite rule: context restriction and surface coercion combined.

Roark et al. (2007) describes these rules as algebraic operations that a computer can transform into two FSTs, one which can read lexical forms as input and one which can read surface forms. These two FSTs are then combined and run simultaneously, resulting in one larger FST which can both generate and analyze words. Mathematically, the larger FST is the logical intersection of the smaller ones. Since regular relations are closed under intersection, FSTs can be used to handle the regular language of partial reduplication, where the number of characters to copy is bounded and finite.

As an example of a rule component, I will describe excerpts from the Wamesa morphological analyzer. Swanson (2019) first defines subsets of the Wamesa alphabet that are necessary for the computer's interpretation of the rules. For example, he specifies which alphabet characters should be interpreted as high vowels, glides, and vowels in the following way:

HighVow = I U i u
Glides = J W j w
Vowel = A E I O U Y
a e i o u y

Using the above groupings, Swanson created two-level rules to account for phonological changes in Wamesa, including the following:

1. "Mid vowel raising"
   e:i <=> _ :u

2. "High vowels -> glides"
   HighVow:Glides <=> .#. _ ;
   :Vowel _ :Vowel ;

In both rules, that which appears the the right of the operator **<=>** specifies the environment or morphotactic constraints. The mid vowel raising rule indicates that when a lexical /e/ appears before a surface [u], the /e/ becomes [i] in the surface realization of a word. The high vowels to glides rule indicates that underlying high vowels are realized as glides in the surface form when located word initially or between two surface vowels.

All in all, Koskenniemi created a framework for researchers to more efficiently handle morphological analysis using computers. Since Koskenniemi's first implementation in 1986, two-level rules have been commonly used in morphological analyzers due to their effectiveness.

2.4 The Helsinki Finite-State Toolkit (HFST)

The Helsinki Finite-State Toolkit (HFST) is a technology used by researchers to build morphological analyzers for various languages. It was first released in 2008 and continues to be an open-source toolkit that is publicly accessible and free to use. Washington et al. (2012, 2014) used the HFST to create morphological analyzers for four Turkic languages. The morphological analyzer for Wamesa (2019) was also built using the HFST. The toolkit consists of a lexicon compiler known as `lexc` and a two-level rule compiler known as `twolc` that are responsible for transforming code into FSTs. In accordance with the KIMMO model, which requires analyzers to have a lexicon component and rule component, the codebase for analyzers built using the HFST must contain a `lexc` file describing a language's lexicon and a `twolc` file containing two-level morphophonological rules.

2.5 The Morphological Analyzer for Wamesa

The morphological analyzer for Wamesa was built as part of the Apertium project, which has the goal of creating translation systems for lesser resourced languages (Washington et al., 2014). The analyzer was built by Daniel Swanson using the HFST, and contains `lexc` and `twolc` files, among others, which work together to define the Wamesa grammar so that the HFST can create FSTs.

The `lexc` file contains information about the morphotactics and lexicon of Wamesa. The file contains morphemes or parts of speech separated according to their morphotactics. Categories include the following: Root, Adjective, VerbPrefix, VerbApplicative, VerbCausative, Determiner, DeterminerNumber, and DemonstrativeRoot (Swanson, 2019).

The following chart shows the sublexicons contained in the lexicon portion of the `lexc` file, along with the number of stems and/or symbols in each sublexicon.[1]

[1]The number of stems (not including punctuation) in the lexicon of the analyzer totals to 1276, but the lexicon is not an exhaustive account of the Wamesa language.

Sublexicon	Number of Stems
AdjectiveRoot	116
AdpositionRoot	2
AdverbRoot	46
Clitics	12
ComplementizerRoot	5
ConjunctionRoot	6
ImperativeRoot	1
InterjectionRoot	5
InterrogativeRoot	8
LocativeRoot	3
NounRoot	660
NumeralRoot	27
ParticleRoot	5
PrepositionRoot	10
PronounRoot	17
ProperNounRoot	5
VerbRoot	294
Punctuation	22

Fig. 2.4: Sublexicons in the `lexc` file (Swanson, 2019)

The `twolc` file contains two-level rules to account for morphological and phonological rules in Wamesa, including mid-vowel raising, cluster simplification, and subject infixation. The file also contains alphabet characters and tags corresponding to those in the `lexc` file. Additionally, alphabet subsets are defined to allow rules to be simpler and clearer. For instance, a correspondence pair can be written the sets HighVow and Glides can be defined as follows:

HighVow = I U i u ;
Glides = J W j w ;
HighVow:Glides

Without the definition of these sets, however, four pairs would need to be created: I:J, U:W, i:j, u:w. Since each rule in the `twolc` file requires one correspondence pair, the absence of defined sets would result in redundant code.

When given an input for word generation, the machine will pass the input through the FSTs created from the `lexc` and `twolc` files. For example, if a user gave the input %[%+2sg%]*pera* to generate the word *puera* 'you cut', the analyzer would know to convert %[%+2sg%] to %{B%}u%> and prefix it to *pera*. Next, the following rules in the `twolc` file would be applied to *%{B%}u%>pera*.

```
"2SG infix 1"
%{B%}:Cx <=> _ u: %>: Cx: ;
 where Cx in Consonants ;

"2SG infix 2"
Cx:0 <=> %{B%}: u: %>: _ ;
where Cx in Consonants ;
```

In reality, these two rules are applied simultaneously, but for the purposes of explanation, the "2SG infix 1" rule would convert *%{B%}u%>pera* to *pupera*, and the "2SG infix 2" rule would delete the second (and original) consonant, resulting in *puera*.

For word analysis, the opposite process occurs, where surface forms are converted to underlying forms and matched with definition in the `lexc` file.

Literature Review

3

3.1 Morphological Analyzers Constructed Using the HFST

In this section, I will describe the work of Washington et al. (2012, 2014), who built morphological analyzers for Turkic languages using the HFST. The researchers demonstrate that the HFST is an effective tool for the needs of a morphological analyzer.

The researchers Washington, Ipasov, and Tyers (2012) built a morphological analyzer for Kyrgyz, a Turkic language classified under the Kypchak and South Siberian subbranches. The Kyrgyz analyzer functions to translate Turkish into Kyrgyz and is also effective in the morphological analysis of Kyrgyz (Washington et al., 2012). Following that, the researchers Washington, Salimzyanov, and Tyers (2014) used the HFST to create morphological transducers for three other Turkic languages, Kumyk, Tatar, and Kazakh, as part of the Apertium project.[1] These languages are all part of the Kypchak branch of the Turkic family but are of different subbranches (Washington et al., 2014). Each of the transducers contain a tagset that accounts for the main parts of speech (eg. noun, verb, adjective) as well as morphological subcategorization, the classification of morphemes into grammatical roles (eg. number, person, possession).

The tagset of the Kyrgyz analyzer consists of 127 tags, 19 of which account for the main parts of speech and 108 of which account for morphological subcategorization. Washington et al. (2012) built the analyzer to account for Kyrgyz's morphotactic and morphophonological processes, including irregular negatives of finite verb forms, irregular [noun + possessive + case] forms, vowel harmony, voicing assimilation, desonorisation, and lenition.

In addition, Washington et al. (2014) detail how developing transducers for the closely-related Kypchak languages[2] took less time since new transducers could be

[1]The Apertium project has the goal of creating translation systems for lesser resourced languages, but the project's resources are available for use on any language (Washington et al., 2014).

[2]Closely-related languages are part of the same branch. However, they can be of different subbranches, which may themselves have subbranches.

based on existing ones. While it took about six months to develop transducers for Kazakh and Tatar with 90% coverage[3] , it only took two weeks to reach the same level of coverage for the subsequently developed Kumyk transducer (Washington et al., 2014). Since Kumyk has similar syntax to both Kazahk and Tatar, the researchers were able to reuse code from the other transducers and use replacement to account for Kumyk's lexicon, and phonological rules were added as necessary. The authors also discuss how they dealt with issues such as ambiguous characters, loanwords, and acronyms, showing that the HFST is a powerful tool that can be used to build and modify analyzers based on the needs of specific languages. As for future work, Washington et al. highlight the feasibility and importance of developing transducers for other closely-related languages.

Both the Kyrgyz and Kypchak transducers were evaluated by their precision and recall when tested on publicly available texts. The Kypchak transducers were run on texts in the categories of news, religion, and encyclopedia entries (Washington et al., 2014) .The Kyrgyz transducer was able to account for about 85% of the words encountered in two publicly available corpora and also had high precision[4] and recall[5] on a verified test set (Washington et al., 2012).

Thus, the use of the HFST proves to be an effective method of creating morphological analyzers, as the Kyrgyz and Kypchak analyzers were able to both generate and recognize a high percentage of words by accounting for nontrivial morphological processes.

3.2 Handling Reduplication in Various Languages

In this section, I will describe how various researchers have handled reduplication in their morphological analyzers for Indonesian, Vietnamese, and Kinyarwanda. Each of these languages have different types of reduplication that also interact with morphological and phonological constraints in the languages.

[3]Here, coverage refers to the percentage of words of a corpus that are accounted for by a given morphological analyzer.

[4]Precision is a metric used to evaluate models. It refers to the number of correctly produced outputs as a proportion of all outputs produced.

[5]Recall is another metric of evaluation. It refers to the number of correctly produced outputs as a fraction of all possible correct answers.

3.2.1 A Two-Level Morphological Analyser for the Indonesian Language

To handle reduplication, the researchers Pisceldo et al. (2008) use the compile-replace feature in `xfst` (Xerox Finite-State Tool), which allows the program to account for arbitrarily complex reduplication. In order to reduplicate a word, compile-replace requires input in the format of `"^["word^2^]"`, where `word` would be replaced by the actual word to be copied. For example, in order to reduplicate the Indonesian word *buku* ‘book’, compile-replace would require the input `"^["buku^2^]"` and transform it into *bukubuku* ‘books’. The authors provide the following diagram to illustrate their implementation:

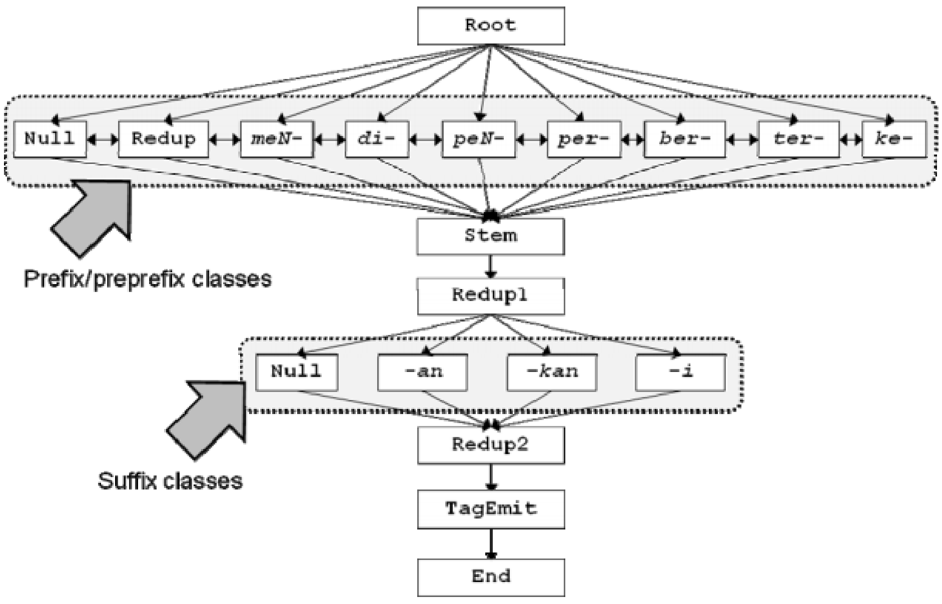


Fig. 3.1: Reduplication in a Morphological Analyzer for Indonesian

Each box represents a state in the FST and are called continuation classes in the `lexc` file. As the arrows show, the process of constructing a word involving reduplication begins at the state labeled "Root", continues to prefix and preprefixes (which may be "Null"), and so on. "Redup1" closes partial and affixed reduplication (when the suffix is not reduplicated), and "Redup2" closes full reduplication. Specifically, these two states will append ˆ]" to create the correct form for the compiler to recognize. "TagEmit" processes lexical tags that were not already handled in the previous states. To generate or recognize a reduplicated word, the analyzer would take one of the paths shown above. This means that as the machine moves from the "Root" state to the "End" state, it can only travel down one of the arrows between each of the intermediate states.

3.2.2 Double Double, Morphology and Trouble: Looking into Reduplication in Indonesian

Although Indonesian has partial, imitative, and full reduplication, Mistica et al. (2009) only focus on full reduplication in this paper. In particular, the authors seek to modify their analyzer to handle productive total reduplication. The authors describe their analyzer, saying it was based on the system built by Pisceldo et al. (2008), which uses the `xfst` and `lexc` tool.

The authors note that full reduplication in Indonesian may be either distributed or reciprocal. For reciprocal reduplication, the doubling of a verb stem must occur before spellout (the application of phonological rules); For distributed reduplication, spellout must occur before doubling (Mistica et al., 2009). In addition to adding a reduplication component, the authors modified their analyzer to have two separate components, each containing morphophonemic and spellout rules, to account for both types of full reduplication. The authors do not provide information about evaluating their modified analyzer.

3.2.3 Finite-State Description of Vietnamese Reduplication

Vietnamese has both partial and full reduplication, and reduplicated words are either two, three, or four syllables (Phuong et al., 2009). In this paper, the authors Phuong, Huyen, and Roussanaly address bi-syllabic reduplication, which accounts for more than 98% of reduplication in Vietnamese. To handle full reduplication, the researchers implemented a minimal sequential string-to-string transducer. In other words, they created an FST that can recognize and generate, with as few states as possible, precisely each of the fully reduplicated words in their analyzer's lexicon. To recognize these 274 words, the resulting FST has 90 states, including 16 final states. It also has 330 transitions, with states having up to 28 transitions out of them. The authors provide an example of a minimal FST handling the words *luôn luôn* 'always', *lù lù* 'silently', *khàn khàn* 'raucous':

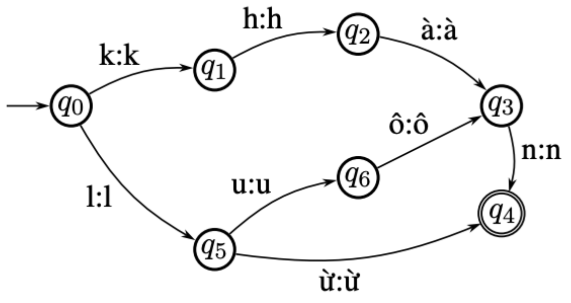


Fig. 3.2: Example of a Minimal FST

Note that the model can only generate and recognize words that are explicitly built into the model, and the list of words is not exhaustive. While the authors do not discuss the ease or difficulty of adding words to their model, the addition would have to be manually done by the researchers, as their model does not have the mechanisms to recognize all reduplicated words or to automatically develop the FST to account for words new to the analyzer.

In order to implement this FST, the authors created a Java software package named vnReduplicator. The authors discuss their use of the package to scan for and recognize bi-syllabic reduplicated words in Vietnamese text. As of writing this paper, the authors did not use their package for other applications but suggest that the software would be useful for spellchecking of reduplicated bi-syllabic words that involve tone changes, which are often incorrectly accounted for by writers. Notably, while the FST built by the researchers can recognize bi-syllabic reduplication in Vietnamese, the FST is not a morphological analyzer which can conduct general morphological analysis on Vietnamese words.

3.2.4 Finite State Solutions For Reduplication In Kinyarwanda Language

Kinyarwanda is a Bantu language which is the official language of Rwanda. Kinywarwanda has both bounded and unbounded reduplication (Muhirwe and Trosterud, 2008), neither of which have been previously accounted for in a morphological analyzer for Kinyarwanda, given that Kinyarwanda is an under-resourced language and that handling reduplication has been a difficult computational task.

For the purposes of this thesis, I will focus on describing how the authors handled partial reduplication, although they also successfully tackled the issues of full word and full stem reduplication. In Kinyarwanda, the reduplicated syllable for partial stem reduplication is of the form CV, VC, or CVN. The authors provide examples of partial reduplication in verbs and nouns with the CV syllable reduplicated. The original words are not provided.

1. *kujejeta* 'to drop/leak'

2. *iseseme* 'nausea'

To account for partial reduplication, they added the following two-level rules to the `twolc` file of their analyzer:

```
Alphabet %+:0 b c d f g h j k l m n
p q r s t v x y z a e i o u;
Sets
C = b c d f g h j k l m n p q r s t
v x y z;
V = a i e o u ;
Rules
"R for realisation as Consonant"
R:CC <=> _ E: %+: CC;
where CC in C;
"E realisation as vowel"
E:VV <=> _ %+: (C:) VV;
where VV in V;
```

They also added the following to the `lexc` file:

```
Lexicon PSPrefix
[Redupli]:RE+ PVRoot;
Lexicon PVRoot
jeta VFinal;
```

Since Kinyarwanda has three types of partial stem reduplication, Muhirwe and Trosterud (2008) created two additional variants of the rules above in order to account for the three types. To account for complex consonants, the authors represented each complex using a multicharacter symbol not used in the lexicon (Muhirwe and Trosterud, 2008). For example, the complex 'ny' was accounted for with the symbol N and the authors added a rule `N -> ny`.

To evaluate their method, the authors conducted positive and negative testing. Positive testing involved words already part of the analyzer's lexicon, and there was 100% accuracy. Negative testing involved giving the analyzer untagged words that were not part of the analyzer's lexicon. While the words used in negative testing were not recognized by the analyzer, the authors note that, given the rules they already wrote, subsequently adding the new words and accounting for reduplication was easy (Muhirwe and Trosterud, 2008). The process involved identifying the part of the new word that needs to be copied and then adding the word to the appropriate sublexicon of the `lexc` file (Muhirwe and Trosterud, 2008). Although the authors did not include quantitative results in their paper, they state that testing 100 known reduplicated forms of different word categories made clear that their methods successfully accounted for all forms of reduplication in Kinyarwanda, including associated complexities specific to the language.

3.3 Additional Approaches to Handling Reduplication

3.3.1 The EQ Operator

Hulden (2009) introduces the use of the EQ operator to handle total and partial reduplication in FSTs. For instance, Hulden gives the example of the word *garadus*, which can be modified into the format <*X*>*gara*<*dus*> (language unspecified). Then, the use of the EQ operator converts that form to <*dus*>*gara*<*dus*>, which corresponds to the final reduplicated form *dusgaradus*. Another example is the word *adus*, which is modified to be <*X*><*adus*> and then turned into <*adus*><*adus*> using the EQ operator. Hulden notes that marking and modifying the input into the form which the EQ operator requires does not have to be done manually and that code can be written to automate certain processes. Ultimately, the EQ operator works to create an exact copy or correspondence between the content in the angle brackets < > where <X> is the segment to be created and copied from the segment of the root word contained within the brackets.

To recognize reduplication, the EQ operator moves across a word to determine which parts match and are copies versus which parts do not match and are not copies (Hulden, 2009). Hulden specifies that this operator can be described as a filter which must be passed during a very specific point of a machine's morphological analysis of a word. This specificity is due to the use of EQ in conjunction with rewrite rules, which must be applied sequentially and whose ordering matters for accurate word generation and analysis. To account for cases where the two reduplicants (copies) are not identical, the EQ operator creates a rule where the surface forms do not need to be equal due to specific phonological constraints. This flexibility means that EQ is not limited to closed lexicon reduplication, since an upper bound on the length of words in a lexicon does not need to be pre-specified. In other words, the EQ operator can generate and recognize reduplicated forms that are not built in to the machine. However, the number of possible reduplicants must still be finite for the EQ operator to work, which means that EQ cannot truly model unbounded reduplication using an FSA, which has a finite number of states. Recognizing this limitation, Hulden suggests the necessity of supplemental methods such as preprocessing, which requires users to modify data before inputting the data to the FSA.

In other words, despite the general perception that 1-way FSTs cannot model productive (open lexicon) reduplication where words not already in a lexicon can be generated, the EQ operator allows 1-way FSTs to produce new reduplicated words. That said, the amount of possibilities that can be generated is still finite, since the form of the reduplicants must still be specified and have an upper bound. For example, if the machine knows that a reduplicant has the form CV and its upper bound is two letters, the machine can generate as many new words which fit those constraints as the set of alphabet characters and the rest of the grammatical constraints in a given language allows. That said, while many possibilities can be generated, some of which may be considered nonsensical or ungrammatical in a language, there is still a finite limit.

3.3.2 Memory Devices

Roark et al. (2007) propose that a memory device could be added to the FST to allow it to handle unbounded reduplication. The device would memorize the characters that have been seen in an input and then match those to subsequent characters which would be from the root word or base. This is similar to how FSTs are coded to memorize input, but the added memory device increases the amount of memory that a 1-way FST would have on its own. Further, Roark et al. write that reduplication should be thought of as two separate components, the first of which models the prosodic constraints and can be handled by finite-state operations, and the second of which (the copying component) requires special treatment (beyond the capabilities of FSTs) to determine if the prefix or suffix matches the base.

As an example of using a memory device, Roark et al. introduce the use of registers, which hold memory, to create finite-state registered automata (FSRAs), which are an extension of FSAs. The number of registers and the amount of memory an FSRA can hold is still finite, but the addition of the registers allow the FST to handle many more possible inputs and outputs. Roark et al. (2007) make clear that without using registers to supplement memory, an FST would need to be much larger in order to account for the many possibilities entailed by reduplication. With the addition of registers, the transition function for an FST must also be modified to allow reading or writing register content and using that content to inform which state the machine should move to. Generally, FSRAs are used to handle non-concatenative morphology, which includes but is not limited to the process of reduplication.

For the FSRA to handle reduplication specifically, the machine writes what was seen in the copy to a register, and the machine reads and compares the register contents to the subsequent root word to determine whether there is a match (Roark et al., 2007). Roark et al. refers to these functionalities as register-write and register-read operations, respectively. However, Roark et al. note that this FSRA approach has limitations: it only handles finite-length copies, and the register operations require that there be an exact match between the characters in the copy and that in the root. Roark et al. note that this restriction is an issue given that copied segments may undergo modification for a number of reasons, such as phonological rules. This is evident in languages across the world, including Dakota, Sye, and Kinande. Roark et al. note that to handle various irregularities, the machine would need to be lexically specific, which would thus defeat the purpose of modifying the FSA into an FSRA.

3.3.3 Skip and Repeat Arcs

As an alternative to memory devices, Roark et al. (2007) suggest adding repeat and skip arcs the FST that would allow the FST to read input in two directions. Specifically, repeat arcs would allow backwards movement on the input tape, while skip arcs would allow skipping over portions of a string. Together, these arcs would allow a word to be copied. This functionality is similar to what Dolatian and Heinz (2018) describe as a 2-way FST, which will be discussed next.

3.4 Handling Reduplication with 2-Way FSTs

This section introduces 2-way FSTs, how they differ from 1-way FSTs, and how they are effective for generating reduplicated words.

3.4.1 Modeling reduplication with 2-way finite-state transducers

The researchers Dolatian and Heinz show that two-way finite-state transducers (2-way FSTs) can handle both partial and total reduplication efficiently in linear time. Dolatian and Heinz discuss various methods that have been attempted previously, such as using context-free grammars (CFGs) and finite-state approximations (supplemented FSTs). However, they note that these methods have only been able to account for unbounded total reduplication (when the number of characters copied varies) with an exponential increase in the number of states in the FST. However, the authors highlight that 2-way FSTs are powerful models for dealing with reduplication because they allow movement back and forth on input but not output (Dolatian and

Heinz, 2018). In contrast, 1-way FSTs do not allow such movement. In this way, 2-way FSTs can account for unbounded copying without a significant increase in the amount of states in the FST. Explaining how the 2-way FST works, Dolatian and Heinz use the terms input tape, output tape, reading head, and writing head: the FST has a reading head that reads the input tape and a writing head that writes to an output tape. After, the machine reader on the input tape can move left or right or stay, but the writing head can only move forward (to the right) (Dolatian and Heinz, 2018). The FST state changes as input is read. The formal definition of a 2-way FST appears below:

Definition: A 2-way, deterministic FST is a six-tuple $(Q, \Sigma_{\ltimes}, \Gamma, q_0, F, \delta)$ such that:

Q is a finite set of states,
$\Sigma_{\ltimes} = \Sigma \cup \{\rtimes, \ltimes\}$ is the input alphabet,
Γ is the output alphabet,
$q_0 \in Q$ is the initial state,
$F \subseteq Q$ is the set of final states,
$\delta : Q \times \Sigma \rightarrow Q \times \Gamma^* \times D$ is the transition function where the direction $D = \{-1, 0, +1\}$.

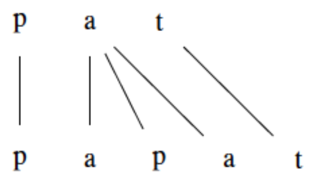
Unlike a 1-way FST that can only recognize and output regular (finite or fully predictable) languages, 2-way FSTs can output non-regular languages, such as the copy language $L_{ww} = \{ww|w \in \Sigma^*\}$ (Dolatian and Heinz, 2018) where w is a character or string of characters and Σ^* is the alphabet of the language. Thus, the 2-way FST is able to output reduplicated words. However, even though the FST can pass through the input multiple times, the run time is still linear based on the length of the word to be copied. (Dolatian and Heinz, 2018).

In the case of total reduplication, the 2-way FST reads a full word from the input tape, moves backwards on the tape until it reaches the beginning of that word, and reads the full word again. Meanwhile, as the machine reads in the forward direction, it is also writing the characters it reads to the output tape (Dolatian and Heinz, 2018).

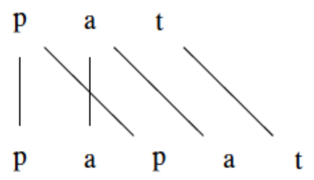
To handle partial reduplication, the 2-way FST reads part of a word, goes back to the location at which it needs to start rereading, and then continues going forward on the input tape. As was the case for total reduplication, the machine writes to the output tape when it moves forward on the input tape.

The authors state that while 1-way FSTs can generate reduplicated words, they do not do so in a truly productive way. The authors are approaching productivity from a psychological standpoint, which entails the generation of new words that have not been used or seen before. However, 2-way FSTs can move in the backwards direction on the input tape in order to actively reread and copy individual characters from the input tape to the output tape. As a result, the origin information, which specifies where in the input an output was produced from, produced by 1-way FSTs is less intuitive. More specifically, various characters in an output may be attributed to a single character from the input. This is a result of 1-way FSTs using memorization of previously seen input in order to reduplicate words. On the other hand, the origin information produced by 2-way FSTs, shows how each output character originates from its corresponding identical character in the input. An example illustrating the differences appears below. The top line of characters is the input, the bottom line is the output, and the lines between them connecting output characters to their origin.

a. Origin information of the 1-way FST



b. Origin information of the 2-way FST



Notice that the 1-way FST attributes the origin of the second <p> and <a> to the input "a" while the 2-way FST attributes all <p>'s to "p" and all <a>'s to "a". Dolatian and Heinz argue that the method that 2-way FSTs handle copying is more intuitive and aligned with how humans would reduplicate word. For those reasons, the authors claim that using 2-way FSTs for reduplication is superior to using 1-way FSTs.

However, while 2-way FSTs are able to create output that reduplicates input, researchers have yet to figure out how to use 2-way FSTs for the morphological analysis of reduplication, which would require being able to take a word with reduplication as input and returning the original root word (Dolatian and Heinz, 2018).

3.4.2 Learning reduplication with 2-way finite-state transducers

In 2019, the researchers Dolatian and Heinz demonstrated the use of a subclass of 2-way FSTs called Concatenated Output Strictly Local functions (C-OSL) functions to succinctly model reduplicative processes. C-OSL functions are based off of Output Strictly Local (OSL) functions (Chandlee et al., 2015) that work with 1-way FSTs to truncate input. While 2-way FSTs and OSL functions already existed, the authors specifically propose the concatenation of OSL FSTs to concisely account for reduplication, in addition to the morphological analysis that 1-way FSTs already handle sufficiently.

C-OSL FSTs are considered a subset of 2-way FSTs since they are more expressive (can account for more computational tasks) than 1-way FSTs yet less expressive than 2-way FSTs (Dolatian and Heinz, 2019a). The hierarchy is as follows: 2-way FST, C-OSL FST, 1-way FST, OSL FST. OSL functions work by truncating input. The C-OSL function works by concatenating OSL functions, as its name suggests. This means that two OSL FSTs are required, and they work together to create an output. Specifically, one FST will truncate the input, while the other FST serves as an identity function and returns the full original input. This effectively handles reduplication since reduplication is defined as word copying and partial reduplication only copies part of a word. For example, for the reduplication of *peya* 'quiet' to *pepeya* 'silent', the first OSL FST will truncate *peya* to produce *pe*, then second FST will produce *peya*, and the concatenation of the output of these two FSTs results in the final output *pepeya*. Reduplication of the second syllable in a word would work in a slightly different way: to produce *kasisiou* 'furious' from *kasiou* 'angry', the first FST would truncate *kasiou* to be *kasi*, and the second FST would read from the end, rather than the beginning of the word, and truncate *uoisak* to be *uois* and then concatenate the remaining *siou* with the result of the first FST, *kasi*, to produce the final result of *kasisiou*. In short, both OSL FSTs will truncate the original input by reading in either the forwards or backwards direction, and then their outputs will be combined to create the final result of the reduplicated word. In the case of total reduplication, both FSTs would output exactly what was given as input, and then those two copies will be concatenated together in the C-OSL FST to create the word resulting from total reduplication.

While Dolatian and Heinz (2019) have determined that C-OSL FSTs work to accurately produce reduplicated words, they have yet to implement them. Additionally, the authors do not discuss their approach in comparison to existing methods of handling reduplication in computational models. The researchers also note that partial reduplication can be modeled with both 1-way and 2-way FSTs and that it is unclear whether one model performs better regarding learning partial reduplication. In other words, whether there is an advantage to using C-OSL FSTs for partial reduplication compared to 1-way FSTs has yet to be determined. In this way, this paper is more theoretical rather than practical in nature. Despite that, Dolatian and Heinz have demonstrated the effectiveness of using the C-OSL function for productive total reduplication, which 1-way FSTs cannot handle.

In addition, however, the authors note that few researchers have demonstrated how a model can learn reduplicative processes, as opposed to having the word forms built into the model already (Dolatian and Heinz, 2019a). Dolatian and Heinz modified the training data for the C-OSL FST learner by marking boundaries between reduplicated copies in the output. They note that this modification is required for the learner to recognize reduplication. While the training data needs to be manually marked by the researchers, the idea is that the model will be able to learn to recognize reduplicated words and generalize to test data which has not been marked nor previously seen.

3.4.3 RedTyp: A Database of Reduplication with Computational Models

In this paper, Dolatian and Heinz (2019) introduce a SQL[6] database called RedTyp which includes 138 reduplicative processes from 91 languages. Specifically, the database is composed of 57 2-way FSTs created by the Dolatian and Heinz that were able to account for 138 reduplicative processes. Additionally, the number of states required for each of the 2-way FSTs they created was significantly less than what would be required by 1-way FSTs accomplishing the same tasks (Dolatian and Heinz, 2019b). The average number of states required for the 2-way FSTs they built was 8.82, with the largest being 28. In contrast, recall that a 1-way FST for Vietnamese required 90 states and 330 transitions to account for one reduplicative process (Phuong et al., 2009). While the database is not a database of all-purpose morphological analyzers but rather of specific reduplicative processes, the significance of the creation of this database is that future researchers will be able to add to, utilize, and reference the FSTs in the database, which account for a wide range of reduplicative processes found in the world's languages. This work is particularly exciting given that researchers working on morphological analyzers for under-resourced languages now have an additional foundation to reference and build off of.

[6]SQL is a programming language used to manage data.

4 Proposal

Since Wamesa does not have total nor productive reduplication, a 1-way FST is sufficient. Additionally, the existing morphological analyzer is built as a 1-way FST using two-level rules. Modifying the software to build a 2-way FST or memory augmented FST would require an overhaul of the current morphological analyzer and would go beyond the necessary computational needs for handling unproductive partial reduplication. Thus, I evaluated the current contents of the `lexc` and `twolc` files in the Wamesa analyzer codebase to inform what symbols, tags, entries, and rules would need to be added or modified in order to account for reduplication.

4.1 Proposal for Handling Reduplication in the Wamesa Analyzer

In this section, I include excerpts of existing code in the analyzer codebase followed by my proposed modifications and additions to the code. The existing code serves as a reference for formatting and ordering guidelines. Brief explanations follow both the excerpts and proposed portions.

The format of the rules I propose below is primarily informed by Muhirwe and Trosterud (2008), whose paper on the Kinyarwanda language details their methods for handling reduplication computationally. Based on my research, I propose that the following code be added to the codebase for Wamesa's morphological analyzer. Steps for developing the proposal included the following: identifying reduplicated words in Wamesa, figuring out the syntax of the codebase written by Swanson, evaluating various methods of handling reduplication in computational models, creating rules for reduplication in Wamesa, determining where in the codebase the proposed rules need to be located and how they would work with existing code in the analyzer.

In the code, the exclamation mark (!) serves to mark comments, which are for the purposes of people reading and trying to understand the code. These comments will be ignored by the computer when it reads the file as they are not necessary for the creation of the FST.

Below I include excerpts of the existing `lexc` file that are relevant to my proposed additions. The notation with percent marks and brackets is used to indicate tags.

```
! Other symbols
%>      ! Morpheme boundary
%{B%}
%{D%}
%{A%}

%[%+2du%]  %[%+2sg%] %[%+3sg%]
%[%-2du%]  %[%-2sg%]  %[%-3sg%]

LEXICON Adjective
AdjectiveRoot ;
ve%<ess%>:ve%> AdjectiveRoot ;

LEXICON VerbCausative
%[%+caus%]:on%> VerbRoot ;
VerbRoot ;

LEXICON VerbInflCaus
%<caus%>%[%-caus%]: VerbObj ;
VerbObj ;

LEXICON VerbPrefix
%[%+2du%]:mur%> VerbApplicative ;
%[%+2sg%]:%{B%}u%> VerbApplicative ;
%[%+3sg%]:%{D%}i%> VerbApplicative ;

LEXICON VerbInfl
%<v%>%<p2%>%<du%>%[%-2du%]: VerbInflAppl ;
%<v%>%<p2%>%<sg%>%[%-2sg%]: VerbInflAppl ;
%<v%>%<p3%>%<sg%>%[%-3sg%]: VerbInflAppl ;

LEXICON AdjectiveInfl
%<adj%>: # ;

LEXICON AdverbInfl
%<adv%>: # ;

LEXICON AdjectiveRoot
```

```
maraba:maraba AdjectiveInfl ;
mararaba:mararaba AdjectiveInfl ;
masabu:masabu AdjectiveInfl ;
masasabu:masasabu AdjectiveInfl ;
nganggau:ŋaŋgau AdjectiveInfl ;
pepeya:pepeya AdjectiveInfl ;

LEXICON VerbRoot
adu:%{A%}du VerbInfl ;
kasiou:k%{A%}siou VerbInfl ;
kavio:kavio NounInfl ;
nganggat:ŋ%{A%}ŋgat VerbInfl ;
peya:peya VerbInfl ;

LEXICON AdverbRoot
saira:saira AdverbInfl ;
```

While reduplicated words are part of the lexicon already, the analyzer does not know that there is a relationship between the root and reduplicated words. In other words, the machine will not be able to generate the reduplicated word given the root word, and the machine will not be able to determine the original root word given the reduplicated version.

Below I include excerpts of the existing `twolc` file that are relevant to my proposed additions. The codebase already contains the following alphabet subsets and symbols.

```
Alphabet
 %{A%}:a %{A%}:0 !to deal with /a/ deletion
 %>:0
 %{B%}:b
 %{D%}:d

 %[%+2sg%]:0 %[%+3sg%]:0 %[%+2du%]:0
 %[%-2sg%]:0 %[%-3sg%]:0 %[%-2du%]:0

Sets
Prefix = %[%+2sg%] %[%+3sg%] %[%+2du%]:0
Suffix = %[%-2sg%] %[%-3sg%] %[%-2du%]:0
Vowels = A E I O U Y
        a e i o u y ;
```

```
Consonants = M N Ŋ P B T D K G V S J R L
           m n ŋ p b t d k g v s j r l ;

Rules
"2SG infix 1"
%{B%}:Cx <=> _ u: %>: Cx: ;
 where Cx in Consonants ;

"2SG infix 2"
Cx:0 <=> %{B%}: u: %>: _ ;
     where Cx in Consonants ;

"3SG infix 1"
%{D%}:Cx <=> _ i: %>: Cx: ;
 where Cx in Consonants ;

"3SG infix 2"
Cx:0 <=> %{D%}: i: %>: _ ;
     where Cx in Consonants ;
```

I include the %[%+2du%] tag for marking 2nd person dual since it is always prefixed to the root word, similar to what happens when the 1st syllable in a word is reduplicated. I include the subject infixation rules since reduplicating the 2nd syllable in a word surfaces in a similar way to infixation, when letters are added within a word. For example,

Subject	Prefix	*ase* 'to swim'	*pera* 'to cut'	*ra* 'to go'
2sg	/bu-/	*bu-ase*	*p<u>era*	*r<u>a*
3sg	/di-/	*di-ase*	*p<i>era*	*r<i>a*

Fig. 4.1: Chart of Subject Infixation (Gasser, 2014)

While the morpheme marking 2nd and 3rd person singular surfaces as an infix when attached to a consonant-initial word, Swanson did not create an infix subset and instead wrote two-level rules for infixation while including the subject marker tags in the prefix and suffix subsets. When a colon does not have a surface form to the right, the surface form is yet to be determined and may vary. My proposed additions follow Swanson's conventions.

The following is code for the `lexc` file that I developed using the implementation of Muhirwe et al. (2008) as a reference. It only includes lexical entries for the Windesi dialect of Wamesa.

```
Multichar_Symbols
%<redup%> ! reduplication

! Other symbols
%{R%}
%{E%}

%[%+redup%] %[%-redup%]

LEXICON Adjective
AdjectiveRedup ;
ve%<ess%>:ve%> AdjectiveRedup ;

LEXICON AdjectiveRedup
AdjectiveRoot ;
%[%+redup%]:%{R%}%{E%} AdjectiveRoot ;

LEXICON Adverb
AdverbRoot ;
%[%+redup%]:%{R%}%{E%} AdverbRoot ;

LEXICON VerbCausative
%[%+caus%]:on%> VerbRedup ;
VerbRedup ;

LEXICON VerbRedup
VerbRoot ;
%[%+redup%]:%{R%}%{E%} VerbRoot ;

LEXICON VerbInflCaus
%<caus%>%[%-caus%]: VerbInflRedup ;
VerbInflRedup ;

LEXICON VerbInflRedup
%<redup%>%[%-redup%]: VerbObj ;
VerbObj ;
```

```
LEXICON AdjectiveInfl
%<adj%>%<redup%>%[%-redup%]: # ;

LEXICON AdverbInfl
%<adv%>%<redup%>%[%-redup%]: # ;

LEXICON AdjectiveRoot
maraba:ma%{R%}%{E%}raba AdjectiveInfl ;
masabu:ma%{R%}%{E%}sabu AdjectiveInfl ;
mase:mase AdjectiveInfl ;
nganggau:ŋaŋgau AdjectiveInfl ;

LEXICON VerbRoot
kasiou:k%{A%}%{R%}%{E%}siou VerbInfl ;
kavio:ka%{R%}%{E%}vio NounInfl ;
nganggat:ŋ%{A%}ŋgat VerbInfl ;
peya:peya VerbInfl ;

LEXICON AdverbRoot
saira:saira AdverbInfl ;
```

In summary, symbols and tags for reduplication along with specifications regarding morpheme ordering need to be added. Instead of VerbCausative going directly to VerbRoot, it should first go to VerbRedup to allow possible reduplication of the verb root. I would remove reduplicated versions of words from the lexicon, since the machine can generate those forms with the new tags. The root word *mase* also needs to be added, as it is currently not in the lexicon. Note that the machine would work slightly differently for reduplication of the first versus the second syllable of a root word. For first syllable reduplication, the machine would prefix %{R%}%{E%} to the root. For example, if a user gives the input %[%+redup%]+*mase*, the machine will prefix %{R%}%{E%} to *mase*. Although Wamesa does not have productive reduplication, this design can account for productive partial reduplication of the first syllable and can be used as a reference for languages with this type of productive reduplication. For second syllable reduplication, the %{R%}%{E%} symbols are included as part of the lexicon entries rather than prefixed. In the case of both first and second syllable reduplication and the presence of a %[%+redup%] tag in the input, the two-level rules I propose below will convert %{R%}%{E%} to be a copy of the CV syllable to be reduplicated. Additional explanation is provided below the following proposed code for the `twolc` file:

```
Alphabet
```

```
 %{R%}:
 %{E%}:
 %[%+redup%]:0 %[%-redup%]:0

Sets
Prefix = %[%+redup%]:0
Suffix = %[%-redup%]:0

Rules
!Reduplication rules
"R1 realization as consonant"
%{R%}:Cx <=> _ %{E%}:  %>:  :Cx;
where Cx in Consonants;

"E1 realization as vowel"
%{E%}:Vx <=> %{R%}:  _ %>:  :Cx  :Vx;
where Vx in Vowel;
where Cx in Consonants;

"R2 realization as consonant"
%{R%}:Cx <=> _ %{E%}:   :Cx;
where Cx in Consonants;

"E2 realization as vowel"
%{E%}:Vx <=> %{R%}:  _  :Cx  :Vx;
where Vx in Vowel;
where Cx in Consonants;
```

The above rules account for the generation and analysis of Wamesa words whose first syllable is reduplicated. A discussion of second syllable reduplication is included in the following section, "Limitations." If a user wants the machine to generate a reduplicated word, a user can input the tag %[%+redup%] followed by the root word. %[%+redup%]:0 indicates that the tag does not appear in the surface form. The symbols Cx, Vx, %{R%}, and %{E%} can be thought of as variables or placeholders whose values will be later determined and may vary. The symbol %> indicates a morpheme boundary which does not surface as anything. R1 and E1 refer to first syllable reduplication; R2 and E2 refer to second syllable reduplication. The only difference is that for R2 and E2, there is not a morpheme boundary between %{R%}%{E%} and the root word.

In the R realization rule, `%{R%}` is being used to represent the first letter in the underlying form of the syllable to be reduplicated. Cx represents a consonant in the Consonants subset. In both instances of Cx, the same consonant is being represented. Thus, the rules indicate that `%{R%}` must surface as the next consonant after `%{R%}`.

In the E realization rule, `%{E%}` is being used to represent the second letter in the underlying form of the syllable to be reduplicated. Vx represents a vowel in the Vowels subset. In both instances of Vx, the same vowel is being represented. Thus, the rule indicates that `%{E%}` must surface as the vowel following the consonant after `%{E%}`. Since the rules are two-level, they are applied simultaneously, allowing the machine to generate reduplicated words. Additionally, the machine will be able to analyze words as reduplicated and identify the original root word.

While the existing `lexc` file contains entries for some reduplicated words, my proposed additions to the analyzer should allow it to generate a reduplicated word, given a reduplication tag and the root word, as well as analyze an input as a reduplication of a specific root word. In this way, the analyzer will be aware of relationships between words rather than treating reduplicated words and their roots as totally distinct from each other.

4.2 Limitations

Limitations and anticipated challenges include inaccurate generation and analysis of second syllable reduplication, translation limitations, and insufficient evaluation: For roots where the second, rather than first, syllable is reduplicated, the analyzer may additionally append a copy of the first syllable in addition to the first, due to morpheme ordering definitions in the `lexc` file. This may not actually be the case, but I am flagging it as a possibility. Additionally, words with second syllable reduplication may not be analyzed properly, as the machine may expect the reduplicant to appear only as a prefix to the root. However, this may not be the case, as the machine should recognized the correspondence pair `%[%+redup%]:%{R%}%{E%}` regardless of whether it is located before or within a root word. Again, I bring up this concern as an untested possibility.

Since reduplicated versions of words are no longer their own entries in the lexicon, the translation capabilities of the analyzer will be limited, as the analyzer will not know the meaning of the reduplicated word, except that it is related to the root word. My research did not cover how to account for this, but future work certainly can.

Since there are very limited examples of reduplication in Wamesa, the analyzer cannot be tested on its ability to generalize. This is not an issue for the morphological analysis of Wamesa but needs to be considered if my approach were to be used as a basis for handling reduplication in other languages that have varied reduplicative processes. In other words, while my approach will work for the words I designed them for, they are not guaranteed to work for words which do not fit the reduplicative pattern of the Wamesa words I have already seen.

Conclusion and Future Work 5

5.1 Conclusion

In summary, the goal of this thesis was to learn about, evaluate, and understand work done so far regarding handling reduplication in computational models, with a focus on morphological analyzers, in order to inform a proposal for handling reduplication in the existing Wamesa analyzer. Primarily, researchers use 1-way FSTs, but the importance and effectiveness of 2-way FSTs for handling reduplication was apparent as well. Methods to supplement or modify 1-way FSTs included the use of memory devices, such as registers, as well as the mathematical EQ operator. A key takeaway regarding reduplication and FSTs is that 1-way FSTs can model bounded reduplication while 2-way FSTs can additionally model unbounded productive reduplication. Additionally, two-level rules are commonly used in the development of morphological analyzers to account for morphological and phonological changes, including reduplication, since two-level rules can be applied simultaneously and succinctly account for lexical and surface form correspondences. Ultimately, I created two-level rules and proposed additional modifications to the Wamesa analyzer that were informed by my thesis research.

5.2 Future Work

The immediate next steps for handling reduplication in the morphological analyzer for Wamesa are to add the proposed code to the `lexc` and `twolc` files of the analyzer codebase, test the modified analyzer on the generation and analysis of the reduplicated words, and evaluate the analyzer based on precision, recall, and coverage. The limitations mentioned in the previous chapter, "Proposal", should also be addressed. While my proposal is informed by my thesis research, I expect that iteration or tweaks to modify and improve my proposal will be necessary. Specifically, there may need to be modifications to content regarding AdjectiveInfl and AdverbInfl in the `lexc` file, as my thesis work did not include researching the notation and requirements for morpheme ordering in the `lexc` file. Even if changes are not necessary to produce accurate results, I would like to acknowledge that my approach is only one of various which can work just as well or even more efficiently. While I would invite researchers to develop and compare approaches for handling reduplication in Wamesa to determine what approach might be best theoretically or computationally, future work should be focused on developing morphological analyzers for other under-resourced, under-documented languages.

Hopefully, the morphological analyzer created by Swanson will continue to be developed by researchers so that it can be used to counter Wamesa's endangered status by augmenting the documentation, awareness, and use of the language. Once the morphological analyzer for Wamesa is developed to account for all morphological processes in the language, researchers can use the analyzer's codebase as a foundation for creating analyzers for closely-related languages. These languages include Biak, Ambai, and Taba, which are also under-resourced and under-documented.

5.2.1 Reduplication in Closely-Related Languages

Below, I briefly discuss reduplication in six languages closely-related to Wamesa as well as how they can be accounted for computationally in comparison to my proposed approach regarding reduplication in Wamesa. In each instance, my description and evaluation is specific to the language being discussed and not a statement regarding any type of reduplication in general.

Ambel has full reduplication, partial reduplication of the first CV cluster, partial reduplication of the first CVC cluster (where the second C is a nasal), none of which are productive (Arnold, 2018). Biak has full reduplication (rare and unproductive) and partial reduplication (frequent) (van den Heuvel, 2006). Neither Ambel nor Biak have the same type of partial reduplication as Wamesa, but reduplication in analyzers for these languages can still be handled with similar two-level rules and lexicon entries. Although Wamesa does not have full reduplication, unproductive full reduplication can be accounted for using 1-way FSTs as long as the root words are part of the analyzer's lexicon and are marked as reduplicable.

Ambai has productive partial reduplication, where the consonant of the stressed syllable is copied and followed by a vowel (/a/, /i/, or /e/) depending on whether the stressed vowel high, low, or back (Silzer, 1983). To account for this type of reduplication, additional phonological two-level rules would be needed. Additionally, a 2-way FST would be needed to account for productivity.

The authors of the Dusner grammar (2012) do not discuss reduplication, but provide an example and mentions that in limited instances, reduplication is used to indicate plurality (Dalrymple and Mofu, 2012). Based on the information provided, I infer that Dusner has limited instances of unproductive full reduplication. These could be accounted for in 1-way FSTs. C-OSL FSTs would work well also but have more than the required computational capabilities.

Iraratu has full reduplication and partial reduplication of the first or second syllable (Jackson, 2014). Both prefixed and suffixed partial reduplication involve vowel deletion in the reduplicated syllable. Based on how prefixed partial reduplication works, I suggest that it be handled as if it were total reduplication and then apply vowel deletion. Two-level rules can be written to handle copying and deletion simultaneously. The author describes reduplication as fairly productive, which means that a 2-way FST would be necessary. Even 1-way FSTs with augmented memory would still only be able to account for a finite number of possibilities.

In the Ma'ya language, nouns are derived from verbs via reduplication of the first and 2nd consonants of the verb stem with an <a> in between (Remijsen, 2001). This reduplication process can be thought of as full reduplication with vowel replacement. The author did not specify whether the process was productive or how common it is.

6 Glossary

A

Affix An affix is a morpheme that is attached to a word. The affix may be a prefix (before), infix (within), suffix (after), or circumfix (around).

B

Bounded copying When the unit to be copied is limited in length and the length is known.

C

Causative Indicates that a subject caused a change of state or an object to do or be something.
Closely-related language Languages which are subbranches of the same branch.
Codebase The collection of code for a project.
Compiler A program which converts code written by humans in programming languages to machine-code, the form which computers actually understand and execute.
Computational complexity A measure of the amount of time and computer memory required to complete a task.
Concatenation The joining of sequences of characters. For example, concatenating 'ab' and 'cd' yields 'abcd' and not 'cdab' or any other combination the characters contained.
Corpus A collection of text.
Coverage Refers to the percentage of words of a corpus that are accounted for by a given morphological analyzer.

E

Essive Expresses a state of being, such as character or temporary location.

F

Finite-state automata (FSA) It consists of input and output alphabets, a set of states, including a start state and accept/final states, and a transition function, which determines which state the machine moves to, given an input and the current state.

Given an input, the FSA can determine whether the input conforms to pre-specified rules. Also known as a finite-state acceptor or a finite-state machine.
Finite-state transducer (FST) A type of FSA which can output certain characters based on the transition function and input characters. Characters may be converted.
Functional components They specify grammatical content, such as case, tense, and agreement, which affect the composition of a word.

G
Grammar The writing system, sounds, and rules that make up a language.

H
Helsinki Finite-State Toolkit (HFST) A set of software used to create transducers for morphological analysis.
Highly-resourced language A language which has many technological resources created for and dedicated to it.

I
International Phonetic Alphabet (IPA) A set of symbols created to document all sounds of human languages.

L
Lexical representation Also known as the underlying representation, this representation indicates grammatical categories such as number, case, tense, and person.
Lexicon The vocabulary of a language.
Lexicon Compiler (lexc) A tool that creates transducers based on code describing a lexicon.

M
Morpheme A part of a word that contributes meaning to the word.
Morphological analysis The process of determining morphemes, what they mean, and how they form words.
Morphological analyzer A computational model which handles morphological analysis.
Morphology The study of morphemes, the meaningful units of words.
Morphophonology The study of morphemes along with sounds.
Morphosyntactic Refers to the ordering of morphemes.

Morphotactics Refers to the constraints regarding the location of morphemes.

O

Open-source Publicly accessible for reference and use.
Orthographic representation The written form of a word.
Orthography The writing system of a language.

P

Partial reduplication When only part of the original word is copied.
Phonology The study of sounds in language.
Precision Precision is a metric used to evaluate models. It refers to the number of correctly produced outputs as a proportion of all outputs produced.
Prosodic constraint A restriction related to prosody.
Prosody Refers to rhythm, stress, and pitch.

R

Recall Recall is a metric of evaluation. It refers to the number of correctly produced outputs as a fraction of all possible correct answers.
Reduplication The process of copying all or part of a word to create another word with related meaning.
Register A component of computers which temporarily holds data.
Regular language A set of words that can be produced according to a finite number of rules.
Rewrite rule A rule used to transform the underlying representation of a word into the surface representation of the word. The application of multiple rewrite rules may be required before the underlying form is fully converted to an accurate surface form.
Root The smallest morpheme in a word with no affixes.

S

Stem A word to which affixes can be added. At minimum, a stem must contain a root, but a stem can already have affixes attached.
Surface form The version of a word which is used when people use the word in writing or speaking.
Syntax The structure of phrases or sentences.

T

Total reduplication When the entire original word is copied.

Two-level rule A rule where the left side indicates the lexical form and the right side indicates the surface form.

U

Underlying form Also known as the lexical representation, this form indicates grammatical categories such as number, case, tense, and person.

Under-resourced language A language with limited, if any, technological resources dedicated to it. Often, under-resourced languages are also under-documented and possibly endangered.

Bibliography

Arnold, Laura Melissa (2018). „A Grammar of Ambel, an Austronesian language of Raja Ampat, west New Guinea". In: (cit. on p. 42).

Chandlee, Jane, Rémi Eyraud, and Jeffrey Heinz (2015). „Output strictly local functions". In: (cit. on p. 30).

Dalrymple, Mary and Suriel Semuel Mofu (2012). *Dusner*. Lincom Europa (cit. on p. 42).

Dolatian, Hossep and Jeffrey Heinz (2018). „Modeling reduplication with 2-way finite-state transducers". In: *Proceedings of the Fifteenth Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pp. 66–77 (cit. on pp. 10, 27–29).

– (2019a). „Learning reduplication with 2-way finite-state transducers". In: *International Conference on Grammatical Inference*, pp. 67–80 (cit. on pp. 30, 31).

– (2019b). „RedTyp: A Database of Reduplication with Computational Models". In: *Proceedings of the Society for Computation in Linguistics* 2.1, pp. 8–18 (cit. on p. 31).

Gasser, Emily (2015). „Wamesa Talking Dictionary, version 1.0". In: (cit. on p. 6).

Gasser, Emily A (2014). „Windesi Wamesa Morphophonology". In: (cit. on pp. 2–5, 35).

Ghomeshi, Jila, Ray Jackendoff, Nicole Rosen, and Kevin Russell (2004). „Contrastive focus reduplication in English (the salad-salad paper)". In: *Natural Language & Linguistic Theory* 22.2, pp. 307–357 (cit. on p. 5).

Henning, Jean C, Theodore A Henning, Mina Sawaki, et al. (1991). „Sane pai ve pir kavavo nana kavo Wondama/Perbendahara'an kata bahasa Wandamen/Wandamen vocabulary (Series B, Publikasi Khusus Bahasa-Bahasa Daerah 8)". In: *Irian Jaya, Indonesia: Universitas Cenderawasih and Summer Institute of Linguistics* (cit. on p. 6).

Hulden, Mans (2009). „Finite-state machine construction methods and algorithms for phonology and morphology". In: (cit. on p. 25).

Jackson, Jason Alexander Johann (2014). „A Grammar of Irarutu, a Language of West Papua, Indonesia, with Historical Analysis". In: (cit. on p. 43).

Karttunen, Lauri and Kenneth R Beesley (2001). „A short history of two-level morphology". In: *ESSLLI-2001 Special Event titled" Twenty Years of Finite-State Morphology* (cit. on p. 12).

Koskenniemi, Kimmo (1983). *Two-level morphology: A general computational model for word-form recognition and production*. Vol. 11. University of Helsinki, Department of General Linguistics Helsinki (cit. on p. 12).

Koskenniemi, Kimmo (1986). „Compilation of automata from morphological two-level rules“. In: *Proceedings of the 5th Nordic Conference of Computational Linguistics (NODALIDA 1985)*, pp. 143–149 (cit. on pp. 12, 13).

Mistica, Meladel, I Wayan Arka, Timothy Baldwin, and Avery Andrews (2009). „Double double, morphology and trouble: Looking into reduplication in Indonesian“. In: *Proceedings of the Australasian Language Technology Association Workshop 2009*, pp. 44–52 (cit. on p. 22).

Muhirwe, Jackson and Trond Trosterud (2008). „Finite state solutions for reduplication in Kinyarwanda language“. In: *Proceedings of the IJCNLP-08 Workshop on NLP for Less Privileged Languages* (cit. on pp. 12, 23, 24).

Oflazer, Kemal (1994). „Two-level description of Turkish morphology“. In: *Literary and linguistic computing* 9.2, pp. 137–148 (cit. on p. 12).

Phuong, Le Hong, Nguyen Thi Minh Huyen, and Azim Roussanaly (2009). „Finite-state description of Vietnamese reduplication“. In: *Proceedings of the 7th Workshop on Asian Language Resources*. Association for Computational Linguistics, pp. 63–69 (cit. on pp. 22, 31).

Remijsen, Albert (2001). *Word-prosodic systems of Raja Ampat languages*. Netherlands Graduate School of Linguistics (cit. on p. 43).

Roark, Brian, Richard Sproat, and Richard William Sproat (2007). *Computational approaches to morphology and syntax*. Vol. 4. Oxford University Press (cit. on pp. 11, 12, 27).

Silzer, Peter James (1983). „Ambai: An Austronesian language of Irian Jaya, Indonesia“. In: (cit. on p. 42).

Swanson, Daniel (2019). *apertium-wad*. `https://github.com/apertium/apertium-wad` (cit. on pp. 12, 16, 17).

van den Heuvel, Wilco (2006). *Biak Description of an Austronesian Language of Papua*. Utrecht: LOT (cit. on p. 42).

Washington, Jonathan, Mirlan Ipasov, and Francis M Tyers (2012). „A finite-state morphological transducer for Kyrgyz.“ In: *LREC*, pp. 934–940 (cit. on pp. 19, 20).

Washington, Jonathan, Ilnar Salimzyanov, and Francis M Tyers (2014). „Finite-state morphological transducers for three Kypchak languages.“ In: *LREC*, pp. 3378–3385 (cit. on pp. 16, 19, 20).