# Annotating Reduplication in Finite-State Morphology

Mans Hulden[1] and Shannon T. Bischoff[2]

[1] University of Arizona
mhulden@email.arizona.edu
[2] Universidad de Puerto Rico Mayagüez
sbichoff@uprm.edu

**Abstract.** This paper is a brief survey of different reduplication-like phenomena in the morphology and phonology of natural languages. The purpose is to suggest a strategy for writers of finite-state grammars to handle reduplication in a simple way. The fundamental observation is that the different kinds of reduplication found across languages could be very easily described in a finite-state system if one were to have access to an operator that enforced the equivalence of finite discontinuous strings at some level of representation.

## 1 Introduction

Reduplication phenomena are a challenge to finite-state developers of language models for primarily two reasons. Firstly, a constraint that different parts of a string be equal in content is, in the general case, not expressible through finite-state means, and in the specific case, where such duplication is restricted to a finite lexicon, tends to lead to an explosion of the number of states in a finite-state system. Second, in the restricted case, asserting the equality of two parts in a string is not easily expressed through existing finite-state operations.

In this paper we shall give a brief overview of the kinds of reduplication found in natural languages, and, assuming a multi-level approach to morphological analysis through composition, suggest a simple notation to include reduplication phenomena in most grammars—that is, if one is willing to accept the first limitation of the growth of the number of states in an entirely finite-state system that models reduplication.

## 2 Reduplication

The classical case of reduplication in morphology and phonology is the phenomenon of complete reduplication—a perennial example is that of Bahasa Indonesia (1) or Axininca Campa (2), where (typically) pluralization is expressed through reduplication of a complete word:

(1)

| Base Form | Reduplication | Gloss |
|---|---|---|
| *buku* | buku-buku | 'book' Pl. |
| *orang* | orang-orang | 'man' Pl. (people) |

[1]

(2)

| Base Form | Reduplication | Gloss |
|---|---|---|
| *kawosi* | kawosi-kawosi | 'bathe' |

[2]

This is a very important type of reduplication, since it appears more or less in every language [3], although not always with an explicit grammatical function as pluralization.[3]

Another often occurring pattern is the phenomenon where a limited amount of material is copied from a stem that may be longer than the reduplicant (Uw Oykangand):

[3] English, for instance, apart from the well known *shm*-reduplication (as in *linguistics-shminguistics*), seems to employ the device of total reduplication as a "contrastive focus" method: "I had a JOB-job once. [as opposed to an academic job]." Corpus studies have revealed that this occurs more often than one would expect: see [4] for examples like the one above, or the Corpus of English contrastive focus reduplications at http://umanitoba.ca/faculties/arts/linguistics/russell/redup-corpus.html.

(3) Base Form Reduplication Gloss
*elbmben* elbmbelbmben 'red'
*algal* algalgal 'straight'
[5]

A special type of this is the case where only a part is reduplicated, with intervening material (Madurese):

(4) Base Form Reduplication Gloss
*garadus* dusgaradus 'fast and sloppy'
*abit* bitabit 'finally'
[6]

Also, reduplication may not always result in identical material—phonological changes may occur that result in that two (or more) sequences are similar, yet not identical, as in this example from Javanese:

(5) Base Form Reduplication Gloss
*bali* bola-bali 'return'
*iba* iba-ibu 'mother'
*udan* udan-udɛn 'rain'
[7, 8]

The more challenging patterns occur when we find dependencies that cross and produce multiple different partial copies of the base, as in Coeur d'Alene:

(6) Base Form Reduplication Gloss
*ɛn'is* ɛ'ɛn'ɛn'is 'little ones went off one by one'
*caq* caqcaqaqɛlipip 'he fell on his back'
[9]

## 3 Finite-state grammars

The prevalent mode of constructing morphological analyzers for natural languages is that of composing a set of finite-state transducers in sequence, finally producing a transducer that maps strings from an analysis to a surface form, and vice versa. Usually the setup is as follows:[4]

```
Morphology .o. Rule1 .o. Rule2 .o. ... .o. RuleN
```

This serial mode of creating surface strings from underlying strings lends itself to a particular way of describing reduplication, namely, enforcing the equality of two parts of a string at *some* level of representation. To return to the previous example of the habitual-repetitive form in Javanese and the word *bola-bali*: presumably, a grammar on the lexical level would describe this word more or less as:

```
bali+HabRep
```

and at a subsequent level, the tag `+HabRep` would be changed to `=bali`, giving *bali=bali* (let us disregard for a moment how this is done). A rule for Javanese is that if the second vowel of the stem is not `a`, the first vowel of the left copy changes from `a` to `o` and from `o` to `ɛ`, and the second vowel changes to `a`, i.e. something like:

```
define CRule a -> o , o -> E || .#. C* _ C* [V-a] ?* = ,,
             [V-a] -> a || .#. C* V C* _ ?* = ;
```

Here, `C` and `V` represent consonants and vowels, respectively.

Backing up to the operation that produces a copy of the root `bali`: there are methods available for performing this kind of a duplication (e.g. the compile-replace algorithm in the xfst toolkit [10]). However, the apparent simplicity of reduplication is somewhat muddied by the complexity of current methods to handle the problem. Working with a serialist description, one where regular relations are

[4] We assume a xfst-like notation here following [10], since that is the tool we used for our testing.

composed sequentially, our survey shows that almost every type of reduplication could be handled if there were some simple notational way to express a constraint about the similarity of strings at some level of representation. In the example above, this kind of a constraint would have to be enforced before composition with `CRule`.

## 4 Enforcing equality

We have experimented with defining an operator `EQ(L,R)` that takes two regular languages as arguments, `L` and `R`, such that it accepts only those languages where material between all instances of `L` and `R` are identical. If such an operator existed, the above example of `bola-bali` could be handled as follows: we could, in the lexical level, define the roots in such a way that they are surrounded by special symbols, which we here call < and > (the choice is arbitrary).

```
<bali>+HabRep
```

then, we would replace instances of `+HabRep` with the language `< NOBR* >`, that is, any string enclosed in the special symbols (that does not itself contain the special symbols). Following this, we would compose this with the operation `EQ(<,>)`, yielding `<bali><bali>`.

Unfortunately, `EQ` is not a finite-state language in the general case, as is well known. But for the cases where we want to constrain equality of a finite number of strings,[5] such an operation would be desirable in a chain of compositions that produce surface forms from lexical forms. This would yield a chain of compositions akin to:

```
Morphology .o. Rule1 .o. ... .o. EQ(<,>) .o. ... RuleN
```

For the purposes of testing the notation, we have approximated `EQ` as follows. First extract the "lower language" (or range) of the transducer composed prior to `EQ`, calling this language `S`. Then:

1. Set n to 2.
2. Extract only the strings from `S` that contain exactly n pairs of `L R`. Then create $L_1 \dots L_n$ from the strings between L and R. If this yields the empty language, go to 6.
3. Intersect $L_1 \cap \dots \cap L_n$, yielding $L$
4. Discard any resulting arcs in $L$ that induce a cyclic path
5. Extract a word list from $L$, add to list $W$. Increase $n$, go to 2.
6. Create from $W$ the language `EQ(L,R) = CO(`$w_1$`) | ... | CO(`$w_n$`) | ~$[L ?* R ?* L ?* R]` where `CO(X)` is `[[~$[L|R] L X R ~$[L|R]]*]`

This is not meant to be an actual practical algorithm, neither efficient nor maximally general (given finite-state limitations), but rather a first approximation to practically test the simplicity of reduplicating grammars, given the availability of an operation `EQ`, or something equivalent.

Where the `EQ` function becomes most useful and transparent is in describing the equality of discontinuous parts of strings, as in the Madurese plural example `dusgaradus`, where the final syllable of the root `garadus` is prefixed. With the `EQ` function, one can, on the lexical level, generate the roots, change `+Pl` tags into a prefixed `< NOBR* >` sequence, surround the final syllable of the root with brackets < and >, and compose the relation with `EQ(<,>)`, giving a sequence of derivations as follows:

```
garadus+Pl
< ... >garadus
< ... >gara<dus>
<dus>gara<dus>      (EQ(<,>) applies here)
dusgaradus
```

Naturally, we will want to have a rule that removes our special symbols after `EQ` is no longer needed.

As a final example, we also implemented the reduplication pattern of Warlpiri, of which some examples are given here:

[5] Or infinite discontinuous strings whose intersection yields a finite number of strings.

(7)

| Base Form | Reduplication | Gloss |
|---|---|---|
| *pakarni* | pakapakarni | 'hit (?)' |
| *wantimi* | wantiwantimi | 'fall' |
| *tiirlparnkaja* | tiitiirlparnkaja | 'split lengthwise' |
| *pangurnu* | pangupangurnu | 'dig' |

[11]

The reduplication pattern works as follows: an instance of `C V (C) (C) V` is copied from the stem, and prefixed. Starting with the lexical level, that contains strings such as: `pakarni+Redup`, we change `+Redup` to a prefixed string `< NOBR* >`, yielding, in this example: `< NOBR* >pakarni`. We then compose this level with the rule:

```
define MarkRedup [C V (C) (C) V] -> %< ... %> || %> _;
```

marking an instance of the prosodic pattern found in the root. Following this, we apply `EQ(<,>)`, and remove the bracket symbols, yielding derivations such as:

```
pakarni+Redup
< ... >pakarni
< ... ><paka>pakarni
<paka><paka>rni
pakapakarni
```

For more complicated patterns, such as the crossing partial reduplications of Coeur d'Alene in example (6), we need to resort to several different kinds of brackets, enforcing `EQ` on multiple occasions. Even so, the notation is transparent and yields grammars that are quite clear.

## 5 Conclusion

There are many advanced methods available for building finite-state networks that encode languages that feature non-concatenative phenomena, including reduplication: e.g. the compile-replace technique [12], adding extra memory to the parsing algorithm [13], or other techniques where reduplication is semantically encoded into the automata [14]. However, for most such phenomena—including perhaps even simple vowel-lengthening—a very compact notation would allow the developer to, at some level of representation, assert that discontinuous parts of a string are equal. Such a function can either be compiled directly into automata, assuming the reduplicants are finite, or, if one is concerned about the size of the transducers, used as the basis for a run-time constraint where the transducer is split into two parts: pre-equality, and post-equality, and equality enforcement happens on the fly as the two are "virtually composed."

## References


1. Macdonald, R.R., Darjowidjojo, S.: A Student's Reference Grammar of Modern Formal Indonesian. Georgetown University Press (1967)
2. Payne, D.L.: The phonology and morphology of Axininca Campa. University of Texas at Arlington, Arlington, TX (1981)
3. Moravcsik, E.: Reduplicative constructions. In Greenberg, J., ed.: Universals of Human Language. Volume 3. Stanford University Press, Stanford, CA (1978) 297–334
4. Ghomeshi, J., Jackendoff, R., Rosen, N., Russell, K.: Contrastive Focus Reduplication in English (The Salad-Salad Paper). Natural Language & Linguistic Theory **22**(2) (2004) 307–357
5. Sommer, B.A.: The shape of Kunjen syllables. In Goyvaerts, D.L., ed.: Phonology in the 1980s. Story-Scientia, Ghent (1981)
6. Stevens, A.: Madurese phonology and morphology. American Oriental Society, New Haven, CT (1968)
7. Kiparsky, P.: The phonology of reduplication (Ms.). Stanford University (1987)
8. Sproat, R.: Computational Morphology. MIT Press, Cambridge, MA (1992)
9. Reichard, G.: Coeur d'Alene. In Boas, F., ed.: Handbook of American Indian Languages. Volume 3. J. J. Augustin, New York (1938) 515–707
10. Beesley, K., Karttunen, L.: Finite-State Morphology. CSLI, Stanford (2003)

11. Nash, D.G.: Topics in Warlpiri grammar. MIT: Ph.D. Dissertation (1980)
12. Beesley, K., Karttunen, L.: Finite-state non-concatenative morphotactics. Proceedings of the 38th Annual Meeting on Association for Computational Linguistics (2000) 191–198
13. Cohen-Sygal, Y., Wintner, S.: Finite-state registered automata for non-concatenative morphology. Computational Linguistics **32**(1) (2006) 49–82
14. Walther, M.: Finite-state reduplication in one-level prosodic morphology. Proceedings of the first conference on North American chapter of the Association for Computational Linguistics (2000) 296–302