A Large-Scale, Computerized Phonological Analysis of Malagasy

Daniel M. Albro

October 17, 2002

1 Goals of this research

- To cover, ideally, the phonology inherent in the entire vocabulary of a reduplicated language
- To be able to check the analysis automatically.
- To think about how such an analysis was developed and see if there are learnability implications.
- To write tools that might be useful for doing such an analysis, or that might be able to produce such an analysis automatically or semi-automatically. The tools might also help in teaching Optimality Theoretic Phonology and in writing up analyses once they are created (or while they are being created).
- To computationally model a subset of Correspondence Theory, at least as it applies to reduplication.
- To look at computational implications for Correspondence Theory analyses.

2 Development of the Analysis

2.1 First Stages—Computational Framework

2.1.1 OTP

The framework I began with as a basis for this work was that of Albro (1998), based on Primitive Optimality Theory (OTP) (Eisner 1997).

- An OTP representation might appear as follows:

son	[	+	+	]	-	-	-	-	-	-
cont	[	+	+	—	+	]	-	[	+	]
$\underline{\text{son}}$	-	-	[	+	+	+	+	]	-	-
$\underline{\text{cont}}$	-	-	-	[	+	]	-	[	+	]
σ	[	+	+	+	+	+	+	—	+	]

It is a gestural score, a table whose rows are tiers representing (loosely) phonological features and whose columns represent (loosely) slices of time and correspondence relations.

- In OTP a (probably infinite) candidate set is represented as a finite state machine, each path through which represents a gestural score of the type given above.
- OTP constraints are weighted finite state machine over the same alphabet as the constraint sets. These constraints accept all strings in the relevant alphabet, but assign differing weights. The constraints come in two primitive families — $A \rightarrow B$ and $A \perp B$, where (loosely speaking) the first mandates the mutual presence of particular symbols in a time slice, and the second forbids it.

2.1.2 Problems with the OTP representation

- The constraints of OTP are not necessarily intuitive.
- Some necessary generalizations are hard to express in OTP constraints without multiplying the number of tiers required for an analysis through the addition of "workpad tiers." Since the representation complexity and generation complexity of OTP grows exponentially (more or less) with the number of tiers, this is not good.
- The representation has two natural implementations as edge labels of finite state machines:
  1. Each possible time slice "+-[]–" is a separate member of the FSM alphabet. This makes for an alphabet size equal to 5^n, where n is the number of tiers. In the implementation used by Albro (which is similar in this respect to Eisner's implementation as well), the alphabet size was in fact more like 2^{5n}. This large alphabet can negatively impact FSM minimization, for example, and it makes writing an MCFG grammar to represent the same candidate set problematic, since there is no easy access to the parts of the time slices.
  2. Each symbol from $\{+, -, |, [,]\}$ has its own FSM edge. Then all FSM paths must be of length mn for some $m \in \mathbb{N}$. This makes for an FSM with an extremely large number of edges. Since FSM intersection and Chart Parsing (more on this later) have orders of growth based on the number of edges in their inputs, this is a bad thing.

—	b	
—	b	. . .
-	-	
-	-	

Table 1: Representation for surface —b, underlying —b

2.1.3 A Brave New Framework

Actually, not so brave. Here is how the current framework differs from the standard OTP one:

1. We reduce the set of edge indicators from $\{[,], |\}$ to simply $\{|\}$.
2. We replace the "+−+−++" time slices with named segments: a, b, c, *etc.*
3. Instead of features being part of the representation, the representation is simply made up of named segments (or autosegments/segment particles, since a "segment" here could represent a floating tone, or a syllable, or part of an affricate contour). Features are then defined as subsets of the set of segments. You can then try to define the segments such that there is one segment for each combination of features, or you can limit yourself to observable output segments plus, optionally, abstract underlying segments. The latter approach could be justified as following what we know about the acquisition of phonology...
4. The representation is similar to the second possibility for OTP. The representation of table 1 might be as shown in figure 1. This makes specification of a reduplication MCFG simple because all of the objects of reference are available in a discrete form.

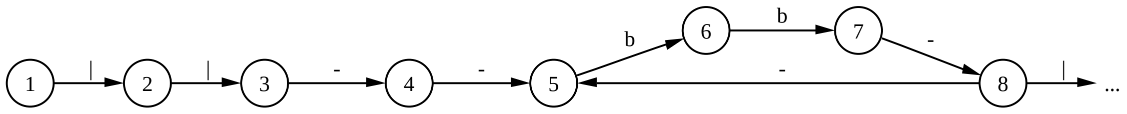

Figure 1: Current Representation

5. Constraints are simply defined here as arbitrary FSMs, with parameters. This allows for:
   (a) More intuitive constraints
   (b) The ability to avoid unnecessary multiplication of tiers.

—	b	—	a	a	—	b	—	a	—
—	b	—	a	a	—	b	—	a	—
—	b	—	a	→**—**←	-	-	-	-	-
B	B	B	B	B	R	R	R	R	R

Table 2: Representation of “ba+RED”

6. The new representation, at least as employed in this particular analysis, has drawbacks as compared to the OTP one in that:

   (a) Some constraints need to be nondeterministic to deal with the lack of an explicit marker for the end of the word. I could easily fix this by adding ’#’ at the end of all representations.

   (b) The OTP representation used a separate tier for each morpheme, which made it easy to refer to them and have one abut the other or even occur inside another. The representation used here is to simply code each morpheme with a different symbol, and not to use the “—” divider symbol between morphemes. The plus side of this approach is that it simplifies rule writing. One of the minuses is illustrated in table 2, where the divider symbol shown in bold face is the locus of a mandatory *SPLITBR violation.

2.2 The Non-Reduplicative Portion

Before discussing how reduplication was handled, we can delve into how the analysis was constructed. The steps were more or less as follows:

1. Tried to gather together a summary of all the patterns found in the Malagasy data. For this I used the following sources: Erwin (1995), Keenan and Razafimamonjy (1995), Keenan and Polinsky (1998), Keenan and Razafimamonjy (1998), Hollanger (1973), and personal communication with Ying Lin and Ed Keenan. One of these days I should try to talk to a native speaker...

2. Started with an initial constraint set representing perfect faithfulness.

3. Added constraints and/or rankings to account for each data pattern in turn. This mostly involves adding new constraints until all of the data for a given pattern are covered.

4. After accounting for a particular data pattern, attempted to determine as much as possible what room for variation there was between rankings that can generate the data patterns covered so far. Also attempted to simplify the analysis by removing unnecessary constraints.

5. Tried periodically to write up the current state of the analysis, with tableaux and Hasse diagrams.

6. Specified new constraint families as needed.

Nothing unusual here. The only unusual thing, perhaps, is my laziness. I ended up writing tools to do as many of these functions as possible automatically. These included:

1. Graphical display/entry/generation by paradigms.
2. Automatic collection of ranking arguments.
3. Automatic production of Hasse diagrams.
4. Automatic calculation of unresolved rankings.
5. Automatic calculation of constraint strata.
6. Automatic constraint ranking given expected outputs (no need for candidates).
7. Automatic detection of useless constraints (very slow, though).
8. Output and generation of tableaux in HTML, ASCII, and LaTeX formats.
9. Generation of entire analysis document with tableaux necessary to justify rankings.
10. Automatic storage of incorrect outputs as candidates.
11. Automatic detection of incorrect or ambiguous outputs.
12. Generation of candidates by varying rankings within known parameters.

The basic constraint families used were as follows:

- Ident (IO, BR, *etc.* and positional variants)
- Faith (same variants as Ident—used for feature as opposed to segment faithfulness)
- Max and variants
- Dep and variants
- *Split (Integrity) (IO, BR, *etc.*)
- *Coalescence (Uniformity) (IO, BR, *etc.*)
- *Sequence, *DoubleSequence, *FinalSequence, *etc.*

- *SequenceIO, *etc.*
- ExtMax—existential faithfulness, sort of.
- * and positional variants thereof.
- *Contour, *ContourIO
- Exists(S)— a segment bearing feature S must exist within the output.
- PreserveRt/LftM—see reduplication section
- Contiguity—this is McCarthy and Prince's O-Contiguity, I think.

2.3 Tour of the Non-Reduplicative Data Patterns

1. For the most part, the output looks like the input. This is the part most non-computerized analyses leave out. Necessitates currently about 22 constraints.
2. Final consonant alternations which appear when weak or pseudoweak stems acquire vowel-initial suffixes. Voiced fricatives, voiceless peripheral fricatives, and some surface [n]'s seem to disappear when not provided with vowel support. The consonants that do not drop become uniformly [ʈɕ], [k], or [n].
3. Nasals assimilate in place to the following consonant.
4. Syllable structure is (N)CV, with sequences sharing place, and contours sharing place and voicing, as well as obeying various sonority sequencing constraints.
5. Consistent with the above, clusters simplify in various interesting ways (Keenan's STOP()).
6. Stress pattern is right-to-left trochaic, with primary stress final, and lapse occurring in pseudoweak and weak stems and possibly also when the antepenult is heavy, if I have the data right. Clash can occur in certain circumstances as well.
7. In hiatus created by suffixation, certain clusters simplify with compensatory lengthening. Elsewhere, different types of clusters simplify, but without compensatory lengthening. The types of hiatus that are avoided differ between prefixing, compounding, and suffixation.
8. [e] may not appear after the final stress of a word.

9. Some weird miscellaneous vowel tricks before deleted consonants: /av/ and /az/ become [i], /iav/ becomes [i], /iaŋ/[1] becomes [i].

10. In (m)an- prefixation the /n/ of the prefix interacts with following consonants differently from other /n/'s of the language.

2.4 Reduplication

2.4.1 Data Pattern

- Stress locations are identical between the base and the reduplicant.
- It's not entirely clear, as Bruce Hayes points out (pc), whether the reduplicant is a suffix (as it is traditionally analyzed), or a pretonic infix. In the first case, the reduplicant is one-to-three syllables, with one, primary, stress on the left-most syllable. In the second analysis, the reduplicant is a left-headed foot, possibly a defective (one-syllable) foot, if there are not enough syllables to create a two-syllable foot. This analysis assumes that the reduplication is suffixing.
- Relativization of reduplicated verbs can create an odd situation where stress locations are no longer identical between the base and the reduplicant; the stress pattern in the base is instead consistent with the unsuffixed form, whereas the reduplicant's stress shifts right, creating a lapse.
- Other than these quirks, reduplication and compounding have the same data pattern.

2.4.2 Computational Background

Correspondence in the OTP framework (and the derived framework used here) is via membership in a so-called time slice. The probably in surface-to-surface correspondence is to bring the two surfaces into a single time slice. I do this by means of a reference tier, or tiers, in which the surface representation we're being faithful to is placed. Note that you could potentially use this mechanism for Paradigm Uniformity and Sympathy as well as for reduplication. The problem in reduplication is that the two surfaces are being computed at the same time.

The solution proposed here is to employ the following structure:

SR	BASE	RED_1
UR	UR_1	UR_2
Ref.	RED_2	—

[1] I model the surface [n]'s that delete without vowel support as /ŋ/.

With this structure we require some mechanism to ensure that the candidate set contains only candidates in which RED_1 and RED_2 are identical.

We can ensure this feature of the candidate set by representing it by a grammar formalism in which it is possible to impose dependencies (*e.g.* identity requirements) between distant sub-elements of the candidate set. The minimally powerful formalism that can do this is a mildly context-sensitive grammar, such as a Minimalist Grammar (Stabler to appear; Stabler 1997), a Multiple Context Free Grammar (Seki *et al.* 1991), a Tree Adjoining Grammar, and some others.

I chose to use MCFGs and developed three or four different types of chart parsers, some valid, some invalid. The current version is a weighted bottom-up chart parser with some adaptations to be specific to a data set where all string lengths are multiples of a given number.

3 Some Implications for Analyses

3.1 Relative ranking of BR versus IR constraints

Dealing with a candidate set that has long-distance dependencies in it is fundamentally more complex, computationally, than dealing with one that merely must be a regular language. In this particular implementation, the chart parser has a worst-case order of growth of something like $O(e^2 + n^8)$, and an average-case order of growth of between $O(e^2 + n^3)$ and $O(e^2 + n^4)$, where n, loosely, is the number of states in the finite state machine being parsed and e is the number of edges. (This isn't very exact). Finite state machine intersection, on the other hand, is on the order of $O(e^2 n^2)$, where e is the number of edges in the machine and n is the number of states. In my implementation, I still have the problem that time spent minimizing the resulting MCFGs is actually overwhelming the time spent intersecting them with constraints. As a practical matter, if you can get the candidate set below say 2000 edges before morphing it into an MCFG, things work pretty well; otherwise, they don't.

The effect of all this is that to have a computationally feasible analysis, you have to try to put BR constraints as low in the ranking as possible in order to avoid introducing the MCFG until the candidate set is compact. This means that wherever possible I have IR constraints outrank BR constraints, in contradiction to the prior ranking given by McCarthy and Prince (1995). Also, I define IR constraints by reference to the underlying and surface tiers rather than the underlying and reference tiers for this reason as well.

3.2 Anchor/Contiguity

The other computational point on which I differ from McCarthy and Prince is in the mechanism that decides what portion of the reduplicating string actually surfaces in the reduplicant. The standard model is to use the ANCHOR constraint in combination with CONTIguity. The Anchor constraint says that the segments at the designated periphery of the base and the reduplicant must be in correspondence. The Contiguity constraint further specifies that there should not be sequences $a, b \ldots c, d$ of underlying segments where a and d are preserved, but $b \ldots c$ are not.

- I'm not happy with the Anchor/Contiguity system for two reasons:

  1. It relies on a gradient version of Contiguity, which is impossible to define in finite state terms without adding a new tier for computational purposes (new tiers can act sort of as stacks, thus increasing the power of the formalism, at the expense of computational complexity). Note that a weaker binary version of contiguity is easy to define (either there is a "deletion gap" between output segments or there is not), but it leads to unusual reduplicants (preserving at both the left and right edges if any gap is necessary).
  2. It relies on identity for a *particular* element (the last or first) rather than something more akin to the earlier "copy from right to left or left to right" rule-based approach. This approach is somewhat fragile, and would not handle the case where the peripheral element of either the base or the reduplicant must be deleted for some reason or another.

- My alternative is easier to compute in finite state terms, and is a bit more flexible:

  1. PreserveRtM(segments, red) or PreserveLftM(segments, red), depending on whether reduplication is suffixing or prefixing. This outputs a violation for every non-preserved element after the first preserved element (for PreserveRt), or for every non-preserved element before the first preserved element (for PreserveLft).
  2. This is outranked by ExtMaxIR(segments), which just says that the reduplicant cannot be empty.

3.3 Third difference

OK, OK, there's a third point of difference, but it's relatively minor — I have not implemented a Linearity (*Metathesis) constraint family because the current representation does not allow for metathesis. This is possible within the framework, however — just allow representations with two underlying levels. In one, the segments are ordered randomly, and in the other the segments are ordered correctly. Constraints other than the Linearity constraint would refer to the disordered underlying form, and the Lineary constraint would require identity between the two representations. This is of course more expensive computationally, but could be tempered by allowing the initial possibility of some limited number of metatheses per word into the candidate set.

4 Future Work

Some of these things will be part of my dissertation, and some won't.

1. Add more data items to the analysis. Right now I have covered Erwin 1995 and the reduplication examples from Keenan and Polinsky 1998. I'd like to add a significant percentage of the dictionary entries before January.

2. Improve the grammar minimization algorithm.

3. Consider an alternative that dispenses with the reference level altogether and uses the known length of the underlying form to create input-specific BR correspondence constraints. This method, if workable, would allow us to get Correspondence Theory reduplication without recourse to non-finite-state methods. It might become somewhat tricky with infixation, however.

4. Infixing reduplication, or reduplication with infixes inside of it. This might be handled by a more sophisticated initial reduplication MCFG, such as one that has BASE+RED followed by RED+BASE (I'm not sure if this sort of infixing reduplication occurs – it would be like ba-bada-dafulo). The kind that just has stuff-¿reduplicant-¿copied stuff can be handled with the current system, except that you might want to have an input representation that does not make the identity of the reduplicant and base specific. That would be a closer fit to McCarthy and Prince anyway, but I've avoided it due to its computational expense. Note that the approach alluded to in (3) above would not be able to cope with a situation where the length of the underlying form of the base is not known at input time, but the current approach would (albeit at some computational expense). Actually, if the location of the base and reduplicant are completely determined by the time the first BR constraint is introduced, the MCFG-related extra expense is nil.

5. Machine learning of reduplication analyses.

6. Improve OTPad, my tool set, and release it to the public. Necessary, or at least desired, improvements include:

   - Minor tweaking with the graphical interface.
   - Allow the user to specify constraint families (right now they're embedded in the code, in one 10,000 line C++ file), and to determine what tiers will exist and what segments may appear on them. The current constraint families are defined for just four tiers: Surface, Underlying, Reference, and Morphological. This makes the representation essentially SPE-like, with no syllable structure, feet, etc.
   - Pick a licensing scheme (GPL, BSD, X, etc.).

- Figure out how to output some sort of Word-friendly tableaux or something.
- Add a help function or something.

A Reduplication Grammar

The following lays out the grammar and parsing method currently used for reduplication.

A.1 Axioms and Goals

The goal item is $[(1, n) : S]$, where n is a final state. The axioms are $[(i, j) : \gamma]$for every edge from state i to state j labeled γ.

A.2 Grammar Normal Form

We'll assume a grammar where right-hand sides are limited to two items, and non-terminals and terminals may not mix in right-hand sides.

A.3 Rules of Inference

Rule 1 is as follows:

$\frac{[(p_1,q_1)...(p_n,q_n):\gamma]}{[(p_1,q_1)...(p_n,q_n):A]}$ if $A \to \gamma \in G$

Rule 2 is as follows:

$\frac{[(p_1,q_1)...(p_{n(A)},q_{n(A)}):B]}{[g((p_1,q_1)...(p_{n(A)},q_{n(A)})):A]}$ if $A \to g[B] \in G$

Rule 3 is as follows:

$\frac{[(p_{11},q_{11})...(p_{1n(B_1)},q_{1n(B_1)}):B_1],[(p_{21},q_{21})...(p_{2n(B_2)},q_{2n(B_2)}):B_2]}{[g((p_{11},q_{11})...(p_{1n(B_1)},q_{1n(B_1)}),(p_{21},q_{21})...(p_{2n(B_2)},q_{2n(B_2)})):A]}$ if $A \to g[B_1, B_2] \in G$ and correct chains are formed.

These are pretty simple. We can get weighted intersection by putting weights in these and summing the weights in the antecedent (plus the rule weight, if any) to get the weight of the consequent. We require that the mapping functions of our grammar not delete any part of the right hand side, so we can reconstruct the grammar by applying g in reverse.

A.4 The Reduplication Grammar

Here's our grammar, in suffixing form:

$$\begin{array}{rcl}
S & \to & g_1[Rd], g_1 \to x_{00}x_{01} \\
 & | & g_1[PfxRd] \\
 & | & g_2[Rd, Fx], g_2 \to x_{00}x_{01}x_{10} \\
 & | & g_2[PfxRd, Fx] \\
PfxRd & \to & g_3[Fx, Rd], g_3 \to (x_{00}x_{10}, x_{11}) \\
Rd & \to & g_4[Rda], g_4 \to (x_{00}, x_{01}) \\
 & | & g_5[Rda, Rd], g_5 \to (x_{00}x_{10}, x_{01}x_{11}) \\
Rda & \to & g_5[Rdb, BR] \\
Rdb & \to & g_6[B, O], g_6 \to (x_{10}x_{00}, x_{01}x_{11}) \\
BR & \to & [b, r] \\
B & \to & [a, a], etc. \\
O & \to & g_7[Oa, Ob], g_7 \to (x_{00}, x_{10}) \\
Oa & \to & g_8[A, U], g_8 \to x_{00}x_{10} \\
Ob & \to & g_8[U, M] \\
A & \to & a, etc \\
M & \to & - \\
Fx & \to & g_9[Fl], g_9 \to x_{00} \\
 & | & g_8[Fl, Fx] \\
Fl & \to & g_8[Oa, MM] \\
MM & \to & g_8[M, Mm] \\
Mm & \to & -
\end{array}$$

And here is the prefixing version of it ($S, PfxRd, Fx, Rd, B, A, M, Oa, Ob, Fl, MM, Mm$ are the same):

$$\begin{array}{rcl}
Rda & \to & g_5[Rdb, RB] \\
RB & \to & [r, b] \\
Rdb & \to & g_{10}[B, O], g_{10} \to (x_{00}x_{10}, x_{11}x_{01}) \\
O & \to & g_7[Ob, Oa]
\end{array}$$

A.5 Grammar/Representation-Specific Speed-ups

Rule 1 applies only if class(p_i) is in the set of acceptable values stored for each terminal-heading non-terminal. In the suffixing grammar given above, A corresponds to class 1, B to class (3,1), U corresponds to class 2, M to class 3, BR to class (4,4). In the prefixing grammar, B corresponds to class (1,3) and RB to class (4,4).

References

Albro, Daniel M., 1998. Evaluation, implementation, and extension of Primitive Optimality Theory. Master's thesis, UCLA.

Eisner, Jason. 1997. Efficient generation in primitive Optimality Theory. In *Proceedings of the ACL.*

Erwin, Sean. 1995. Quantity and moras: An amicable separation. In *The Structure of Malagasy I*, ed. by Matthew Pearson and Ileana Paul, number 17 in UCLA Occasional Papers in Linguistics, 2–30. UCLA Linguistics Department.

Hollanger, Frederick S. 1973. *Diksionera/Malagasy-Englisy*. Lutheran Press and the American Cultural Center.

Keenan, Edward L., and Maria Polinsky. 1998. Malagasy (Austronesian). In *The Handbook of Morphology*, ed. by Andrew Spencer and Arnold M. Zwicky, chapter 28, 563–623. Blackwell.

——, and Jean Paulin Razafimamonjy. 1995. Malagasy morphology: Basic rules. In *The Structure of Malagasy I*, ed. by Matthew Pearson and Ileana Paul, number 17 in UCLA Occasional Papers in Linguistics, 31–48. UCLA Linguistics Department.

——, and ——. 1998. Reduplication in Malagasy. In *The Structure of Malagasy III*, UCLA Working Papers in Syntax and Semantics. UCLA Linguistics Department.

McCarthy, John, and Alan Prince. 1995. Faithfulness and reduplicative identity. In *Papers in Optimality Theory*, ed. by J. Beckman, S. Urbanczyk, and L. Walsh, number 18 in University of Massachusetts Occasional Papers, 259–384. UMass, Amherst: GLSA.

Seki, Hiroyuki, Takashi Masumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science* 88.

Stabler, Edward P. 1997. Derivational minimalism. In *Logical Aspects of Computational Linguistics*, ed. by Christian Retoré, number 1328 in Lecture Notes in Computer Science, 68–95. NY: Springer-Verlag.

—— to appear. Minimalist grammars and recognition. *Journal of Language and Computation* .