Primitive Operations in Phonology

Bridget Samuels

Harvard University

UMD - May 20, 2009
My focus: the nature of phonological representations & operations, and how phonology is situated with respect to the linguistic module and cognition more generally. I attempt to provide a phonological companion to, e.g., Hornstein (2009) and Hornstein & Pietroski (To appear).

Today’s goal: develop a theory of ‘generalized SEARCH and COPY,’ uniting the representations of Raimy (1999, et seq.) with the operations of Mailhot & Reiss (2007) and extending this approach to all of (morpho)phonology.
**Introduction**

- **My focus:** the nature of phonological representations & operations, and how phonology is situated with respect to the linguistic module and cognition more generally. I attempt to provide a phonological companion to, e.g., Hornstein (2009) and Hornstein & Pietroski (To appear)

- **Today’s goal:** develop a theory of ‘generalized SEARCH and COPY,’ uniting the representations of Raimy (1999, et seq.) with the operations of Mailhot & Reiss (2007) and extending this approach to all of (morpho)phonology.
Poeppel (2005):

“Linguists... owe a decomposition (or fractionation) of the particular linguistic domain in question... into formal operations that are, ideally, elemental and generic. The types of computations one might entertain, for example, include concatenation, comparison, or recursion. Generic formal operations at this level of abstraction can form the basis for more complex linguistic representation and computation.”
Starting point

- **Reduplication is affixation** (Marantz 1982). Both are driven by the need to find a host for a newly-introduced morpheme. Each time a string enters the phonological workspace, before anything else happens, it must be combined with the string which is already present.

- Raimy (1999, et seq.) establishes a directed graph notation for phonological representations, which are conceived of as strings of segments ordered by precedence relationships.

/kæt/ is shorthand for: 

| # → k → æ → t → % |

or as ordered pairs:

| (#, k), (k, æ), (æ, t), (t, %) |
Reduplication is affixation (Marantz 1982). Both are driven by the need to find a host for a newly-introduced morpheme. Each time a string enters the phonological workspace, before anything else happens, it must be combined with the string which is already present.

Raimy (1999, et seq.) establishes a directed graph notation for phonological representations, which are conceived of as strings of segments ordered by precedence relationships.

/kæt/ is shorthand for: 

\[ \# \rightarrow k \rightarrow æ \rightarrow t \rightarrow % \]

or as ordered pairs:

\[ (\#, k), (k, æ), (æ, t), (t, %) \]
Representations must be flat for the S-M system.

\[ \therefore \text{3-D or ‘non-asymmetric’ structures (loops) must be repaired} \]

\[ \# \rightarrow k \rightarrow \æ \rightarrow t \rightarrow \% \quad /kætkæt/ \]

\[ \# \rightarrow p \rightarrow u \rightarrow k \rightarrow \% \quad /puhk/ \]
Reduplication
Morphology may direct the insertion of a new precedence relationship which creates a ‘backward’ loop in the string

Add \((t, k)\): \[
\# \rightarrow k \rightarrow æ \rightarrow t \rightarrow % \quad = \quad /kætkæt/
\]
Affixation & Templatic Morphology

Morphology may also create a ‘forward’ loop

Add \((X, Z), (Z, Y)\):

\[
\# \rightarrow X \rightarrow Y \rightarrow % = XZY
\]

\[
\begin{align*}
\# & \rightarrow w \rightarrow a \rightarrow n \rightarrow t \rightarrow % \\
e & \rightarrow d \rightarrow %
\end{align*}
\]
Subtractive Morphology
‘Jump links,’ or forward loops which skip one or more lexical segments, cannot be ruled out without additional stipulations (Gagnon & Piché 2007, Gagnon 2008)

Add (o, %): # → g → o → l → o → n → %
Metathesis
Halle (2008) adds metathesis to the list of processes which can be described in these terms.

Add: (#, B), (A, C), C, A):  

\[
\# \rightarrow A \rightarrow B \rightarrow C \rightarrow \% = BCA
\]
Problem: loops can’t be just anywhere. Typology established in Samuels (2009), §4.3.1-3: \{first, second, stressed, penult, last\} element of type \{X, C, V, foot\}.

Solution: use a search algorithm with a limited set of parameters such as that of Mailhot & Reiss (2007)
Search \((\Sigma, \varsigma, \gamma, \delta)\)

1. Find all \(x\) in \(\Sigma\) subsumed by \(\varsigma\) and index them:
   \(\varsigma_0, \varsigma_1, \ldots, \varsigma_n\)

2. For each \(i \in \{0, \ldots, n\}\):
   (a) Proceed from \(\varsigma_i\) through \(\Sigma\) in the direction \(\delta\) until an element
       subsumed by \(\gamma\) is found
   (b) Label this element \(\gamma_i\)

3. Return all pairs of coindexed standards and goals, \((\varsigma_i, \gamma_i)\)

Copy \((\Sigma, \varsigma, \gamma, C)\)

Identify \(\alpha F\) on \(\gamma_i\) and assign \(\alpha F\) to \(\varsigma_i\) if the set of conditions \(C\) on \(\gamma_i\)
are satisfied
Wolof [ATR] harmony (M&R 38)

a. toxi-lEEEn [toxileen] ‘go and smoke’ (imper.)
b. tɛkki-lEEEn [tɛkkileen] ‘untie’ (imper.)
c. seen-uw-OOn [seeunuwoon] ‘tried to spot’
d. tɛɛr-uw-OOn [tɛɛruwoon] ‘welcomed’

SEARCH $\delta = L$ for $\gamma$ specified [-HIGH, $\alpha$ATR]
COPY [$\alpha$ATR] back to $\varsigma$. 
To apply SEARCH & COPY to morphophonological anchoring:

- Affixhood means lacking #, %, or both. This means there is a ‘sticky end’ (ς) on the affix which enables it to concatenate with another string.
  \[ ς_i \rightarrow e \rightarrow d \rightarrow % \]

- Divorce ς from the beginning point of SEARCH (call it β; β = #, %, or \( γ_{n-1} \))

- COPY places \( γ_i \) into a precedence pair: (ςᵢ, e)
To apply SEARCH & COPY to morphophonological anchoring:

- Affixhood means lacking #, %, or both. This means there is a ‘sticky end’ (ς) on the affix which enables it to concatenate with another string.
  \[ ς_i \rightarrow e \rightarrow d \rightarrow % \]

- Divorce ς from the beginning point of SEARCH (call it β; β = #, %, or γ_{n-1})

- COPY places γ_i into a precedence pair: (ς_i, e)
To apply SEARCH & COPY to morphophonological anchoring:

- Affixhood means lacking #, %, or both. This means there is a ‘sticky end’ (ς) on the affix which enables it to concatenate with another string.
  \[ \varsigma_i \rightarrow e \rightarrow d \rightarrow \% \]

- Divorce \( \varsigma \) from the beginning point of SEARCH (call it \( \beta \); \( \beta = \# \), %, or \( \gamma_{n-1} \))

- COPY places \( \gamma_i \) into a precedence pair: \( (\varsigma_i, e) \)
Suffixation

- $\Sigma$ (string in the active workspace): 
  $\# \rightarrow w \rightarrow a \rightarrow n \rightarrow t \rightarrow %$

- $\varsigma$ (initiator of SEARCH): $\varsigma_i \rightarrow e \rightarrow d \rightarrow %$

- $\gamma$ (target of SEARCH): First X

- $\delta$ (direction of SEARCH): L (i.e., beginning at %)

- COPY $\gamma_i$ to $\varsigma_i$.

- $\# \rightarrow w \rightarrow a \rightarrow n \rightarrow t \rightarrow %$

  $e \rightarrow d \rightarrow %$
Infixation

For infixes, *two* applications of SEARCH, with $\gamma_i = \beta_j$.

Budukh durative (Yu 2007:103)

a. čošu čo-r-šu ‘to stab (downwards)’
b. saq’a sa-r-q’a ‘to die’
c. sa?ar sa-r-?ar ‘to become dry’

ς (initiator of SEARCH): $\varsigma_i \rightarrow r \rightarrow \varsigma_j$

γ (target of SEARCH): $\gamma_i = $ First V; $\gamma_j = $ First X

δ (direction of SEARCH): R

β (beginning point of SEARCH): $\beta_i = \#; \beta_j = \gamma_i$

$\# \rightarrow \check{c} \rightarrow o \rightarrow \check{s} \rightarrow u \rightarrow \%$

$\rightarrow r$
Subtractive morphology

Tohono O’odham (Zepeda 1983, Gagnon & Piché 2007)

Imperfective    Perfective

a. hiːnk      hiːn     ‘bark(ed)’
b. ñeid        ñei      ‘see/saw’
c. golon       golo     ‘rake’

ς (initiator of SEARCH): $ς_i \rightarrow ς_j$

γ (target of SEARCH): $γ_i$: Second X; $γ_j$: %

δ (direction of SEARCH): $δ_i$: L; $δ_j$: R

β (beginning point of SEARCH): $β_i$: %; $β_j$: $γ_i$

# → g → o → l → o → n → %
In the case of reduplication, the affix enters with two sticky ends. The second SEARCH can begin at #, %, or $\gamma_i$.

**English shm-reduplication**

- $\varsigma$ (initiator of SEARCH): $\varsigma_i \rightarrow sh \rightarrow m \rightarrow \varsigma_j$
- $\gamma$ (target of SEARCH): $\gamma_i = \text{First X}; \gamma_j = \text{First V}$
- $\delta$ (direction of SEARCH): $\delta_i = \text{L}; \delta_j = \text{R}$
- $\beta$ (beginning point of SEARCH): $\beta_i = \%; \beta_j = \#$

$#$ → $f$ → $a$ → → $n$ → $c$ → $y$ → $\%$  

shm
Reduplication

Kamaiurá aspectual reduplication (Yu 2007:111)

*Singular*  *Plural*

a. omokon  omoko-moko-n  ‘he swallowed it (frequently)’
b. ohuka  ohuka-huka  ‘he (kept on) laughing’
c. jeumirik  jeumiri-miri-k  ‘I tie up (repeatedly)’

ς (initiator of SEARCH): $\varsigma_i \rightarrow \varsigma_j$

γ (target of SEARCH): $\gamma_i = \text{First V}; \gamma_j = \text{Second C}$

δ (direction of SEARCH): $\delta$: L

β (beginning point of SEARCH): $\beta_i = %; \beta_j = \gamma_i$

# $\rightarrow o \rightarrow m \rightarrow o \rightarrow k \rightarrow o \rightarrow n \rightarrow %$
Further consequences

**Corollary**: loop direction is epiphenomenal.

- \( \varsigma \) (initiator of SEARCH): \( \varsigma_i \rightarrow \varsigma_j \)
- \( \gamma \) (target of SEARCH): \( \gamma_i = \hat{V}; \gamma_j = \text{Second } V \)
- \( \delta \) (direction of SEARCH): \( R \)
- \( \beta \) (beginning point of SEARCH): \( % \)

\[ 
\# \rightarrow C \rightarrow V \rightarrow C \rightarrow V \rightarrow C \rightarrow \hat{V} \rightarrow \% \\
\# \rightarrow C \rightarrow \hat{V} \rightarrow C \rightarrow V \rightarrow C \rightarrow V \rightarrow \% 
\]
From loops to strings

The linearized output (see Fitzpatrick (2006)):
- Takes the shortest path through the graph (ECONOMY)
- Realizes as many precedence relations as possible (COMPLETENESS)
- Realizes m-links in preference to lexical links
- Takes the shortest loops first (SHORTEST)
From loops to strings

Same results, no constraints: Idsardi & Shorey’s (2007) modified version of Dijkstra’s shortest path algorithm (Dijkstra 1959).

To linearize:
(#, k), (k, æ), (æ, t), (t, %)

Vertices: {#, k, æ, t, %}

Initial queue:
1) # → k
2) k → æ
3) æ→ t
4) t → %
Linearizing (#, k), (k, æ), (æ, t), (t, %)

Step 1 (begin at #)
Traverse path: # → k
Output: # → k
New queue:
1) k → æ
2) æ→ t
3) t → %
4) # → k
Linearizing (#, k), (k, æ), (æ, t), (t, %)

Step 2 (begin at /k/)
Traverse path: k → æ
Output: # → k → æ
New queue:
1) æ → t
2) t → %
3) # → k
4) k → æ
Linearizing (#, k), (k, æ), (æ, t), (t, %)

Step 3 (begin at /æ/)
Traverse path: æ→ t
Output: # → k → æ→ t
New queue:
1) t → %
2) # → k
3) k → æ
4) æ→ t
Linearizing (#, k), (k, æ), (æ, t), (t, %)

The algorithm then reaches %:

**Step 4 (begin at /t/)**

Traverse path: t → %

Output: # → k → æ → t → %

Algorithm halts.

Since each vertex was the starting point for only one path, the order of the statements in the queue did not actually matter; any ordering of statements would have yielded the same output. But this is not the case when there are loops.
Linearizing (t, k), (#, k), (k, æ), (æ, t), (t, %)

If we add (t, k), to the bottom of the queue, nothing changes. But if we add (t, k) above (t, %), everything proceeds as above until Step 4:

Step 3 (begin at /æ/)
Traverse path: æ→ t
# → k → æ→ t
New queue:
1) t → k
2) t → %
3) # → k
4) k → æ
5) æ→ t
Step 4 (begin at /t/)

Traverse path: t → k
# → k → æ → t → k

New queue:
1) t → %
2) # → k
3) k → æ
4) æ → t
5) t → k
Step 5 (begin at /k/)

Traverse path: k → æ

# → k → æ→ t → k → æ

New queue:

1) t → %
2) # → k
3) æ→ t
4) t → k
5) k → æ
**Step 6 (begin at /æ/)**

Traverse path: \(æ\rightarrow t\)

# → k → æ → t → k → æ → t

New queue:

1) \(t \rightarrow %\)
2) \(# \rightarrow k\)
3) \(t \rightarrow k\)
4) \(k \rightarrow æ\)
5) \(æ \rightarrow t\)
Finally, the algorithm hits %:

**Step 7 (begin at /t/)**

Traverse path: t → %

Output: # → k → æ→ t → k → æ→ t → %

Algorithm halts.
More on linearization

- If precedence relations established by SEARCH and COPY are always added to the top of the queue, we get an asymmetric string which realizes m-links in preference to lexical material, and is economical. (No loops in the lexicon: Gagnon (2007), contra Fitzpatrick (2006))

- In Samuels (2009), §4.3.6 I discuss the evidence from Lushootseed for SHORTEST and propose an alternative.

- I also discuss over-/under-application and argue linearization applies immediately upon concatenation (cf. Raimy (2000)).
More on linearization

- If precedence relations established by SEARCH and COPY are always added to the top of the queue, we get an asymmetric string which realizes m-links in preference to lexical material, and is economical. (No loops in the lexicon: Gagnon (2007), contra Fitzpatrick (2006))

- In Samuels (2009), §4.3.6 I discuss the evidence from Lushootseed for SHORTEST and propose an alternative.

- I also discuss over-/under-application and argue linearization applies immediately upon concatenation (cf. Raimy (2000)).
More on linearization

- If precedence relations established by SEARCH and COPY are always added to the top of the queue, we get an asymmetric string which realizes m-links in preference to lexical material, and is economical. (No loops in the lexicon: Gagnon (2007), contra Fitzpatrick (2006))

- In Samuels (2009), §4.3.6 I discuss the evidence from Lushootseed for SHORTEST and propose an alternative.

- I also discuss over-/under-application and argue linearization applies *immediately* upon concatenation (cf. Raimy (2000)).
As early as *SPE*, it was recognized that rule application could be seen as a search plus modification procedure (see also Mailhot & Reiss (2007:30)):

“To apply a rule, the entire string is first scanned for segments that satisfy the environmental constraints of the rule. After all such segments have been identified in the string, the changes required by the rule are applied simultaneously.” (Chomsky & Halle 1968:344)
Generalized Search & Copy

- **Search** simplifies rules and preserves important insights from autosegmental phonology while streamlining representations.

- \( V \rightarrow [+F] / \_ C_0 \# \) stands for an infinite set of simultaneously-applying rules:
  
  a. \( V \rightarrow [+F] / \_ \# \)
  b. \( V \rightarrow [+F] / \_ C \# \)
  c. \( V \rightarrow [+F] / \_ CC \# \)
  d. \( V \rightarrow [+F] / \_ CCC \# \)
  e. \( V \rightarrow [+F] / \_ CCCC \# \)
  
  : 

- I agree with Odden (1994) and others since the 1970’s that this infinity should be re-interpreted as *locality*. In recent years, this has been achieved representationally, (i.e., feature geometry and autosegmental tiers).
Adopting **SEARCH** allows us to maintain this basic result, but procedurally rather than representationally.

Treating [+F] as a highly abstract affix allows the following:

- $\varsigma$ (initiator of **SEARCH**): [+F]
- $\gamma$ (target of **SEARCH**): First V
- $\delta$ (direction of **SEARCH**): L
- $\beta$ (beginning point of **SEARCH**): %
- **COPY** $\varsigma$ to $\gamma$

No matter how many consonants come between % and the last vowel in the word, **SEARCH** will converge on the correct target.
Generalized Search & Copy

Now, rather than COPY adding a feature \textit{from} the target (\(\gamma\)) \textit{to} the initiator (\(\varsigma\)), COPY applies the other way around. Thus, COPY must be made bidirectional:

\textbf{COPY algorithm (bidirectional version)}

Identify \(\alpha F\) on \(\gamma_i\) and assign \(\alpha F\) to \(\varsigma_i\) if the set of conditions \(C\) on \(\gamma_i\) are satisfied \textbf{or}.

Identify \(\alpha F\) on \(\varsigma_i\) and assign \(\alpha F\) to \(\gamma_i\) if the set of conditions \(C\) on \(\gamma_i\) are satisfied.

I add a third primitive operation, DELETE, which interacts with SEARCH and COPY to create the typology of feature-filling vs. feature-changing and pure spreading vs. spread-and-delink.
I adopt the distinction between Path and F-element rules from Archangeli & Pulleyblank (1994):

- **Parameter:** *Function*
  Values: \{INSERT, DELETE\}

- **Parameter:** *Type*
  Values: \{PATH, F-ELEMENT\}

- **Parameter:** *Direction*
  Values: \{LEFT-TO-RIGHT, RIGHT-TO-LEFT\}

- **Parameter:** *Iteration*
  Values: \{ITERATIVE, NON-ITERATIVE\}
Direction corresponds directly to the $\delta$ parameter on SEARCH.

Type distinguishes spreading (PATH) from insertion (F-TYPE) rules. For me, whether $\varsigma$ is in $\Sigma$ or not.

Function corresponds roughly to my distinction between COPY and DELETE operations, but deletion of a segment or string can also come from forward loops.

Iteration is discussed in Samuels (2009), §4.4.4
Conclusions

- The primitive operations **SEARCH, COPY, & DELETE** which I have introduced here, and which are explicated in greater detail in Chapter 4 of my thesis, work in concert to produce the range of attested phonological and morphophonological processes.

- The result is a procedural theory, similar in some respects to Autosegmental Phonology but with much slimmer representations.

- Future directions: experiments on the acquisition of phonological rules, particularly how children generalize and how anchors relate to other cognitive mechanisms, e.g., positional memory; comparisons between phonological and syntactic operations.
Conclusions

- The primitive operations SEARCH, COPY, & DELETE which I have introduced here, and which are explicated in greater detail in Chapter 4 of my thesis, work in concert to produce the range of attested phonological and morphophonological processes.

- The result is a procedural theory, similar in some respects to Autosegmental Phonology but with much slimmer representations.

- Future directions: experiments on the acquisition of phonological rules, particularly how children generalize and how anchors relate to other cognitive mechanisms, e.g., positional memory; comparisons between phonological and syntactic operations.
Conclusions

The primitive operations **SEARCH, COPY, & DELETE** which I have introduced here, and which are explicated in greater detail in Chapter 4 of my thesis, work in concert to produce the range of attested phonological and morphophonological processes.

The result is a procedural theory, similar in some respects to Autosegmental Phonology but with much slimmer representations.

Future directions: experiments on the acquisition of phonological rules, particularly how children generalize and how anchors relate to other cognitive mechanisms, e.g., positional memory; comparisons between phonological and syntactic operations.
Conclusions

Thank you!

http://people.fas.harvard.edu/~bdsamuel


