Calculating Metrical Structure

William J. Idsardi, University of Delaware

May 5, 2005

1. Introduction

This chapter has four main goals: (1) to revise and further simplify the Simplified Bracketed Grid (SBG) theory of metrical computations outlined in Idsardi 1992 and Halle and Idsardi 1995, (2) to relate the parameters of the revised system directly to Finite State Automata (FSAs) giving a clear, effective and efficient computational foundation to the theory, (3) to present some short case studies illustrating interesting aspects of the revised theory, and (4) to compare the automata in the present theory with those constructed using recent Optimality Theory (OT; Prince and Smolensky 2004, etc.) proposals for stress (McCarthy 2003, Hyde 2002, etc.) coupled with Karttunen's 1998 proposals for finite-state compilation of OT grammars.

2. Rules and Machines

The revised SBG theory is largely a response to questions posed by Nigel Fabb during his ongoing collaboration with Morris Halle applying SBG systems to the metrical scansion of poetry (see Fabb 2002, etc.). Nigel asked how the parameterized SBG rules actually worked to scan across a form, and my answer was (rather flippantly) that they, like other phonological rules, could be compiled into finite-state automata, a result known at least since the early 1970s (Johnson 1972, Howard 1973). Formulating the rules as FSAs has certain practical advantages, as many modern (and not-so-modern) programming languages and tools (such as Sed, Awk, Perl, Ruby, JavaScript, etc.) offer regular expression facilities that can be used to implement FSAs.
fairly easily. It is particularly fortuitous that the excellent Xerox software for compiling and manipulating FSAs is finally available for a very reasonable cost (Beesley and Karttunen 2003). This now makes it possible to quickly build and test even very large FSAs and compare different grammars for their effectiveness and efficiency (see in particular section 4, below). Space restrictions prevent me from providing anything like a comprehensive introduction to the use of FSAs in the understanding the application of phonological rules. There is a good introduction to the general topic in Kenstowicz & Kisseberth 1977:188ff and in Kenstowicz & Kisseberth 1979:326ff. Textbooks on mathematical linguistics, such as Gross 1972 and Partee, ter Meulen and Wall 1990 or introductions to automata theory, such as Hopcroft and Ullmann 1979 offer further information on FSAs; the Beesley and Karttunen book is indispensible.

The SBG theory follows Halle & Vergnaud 1987 in viewing the calculation of prosodic structure as being governed by a system of parameterized rules. By this we mean that there this is a set of ordered rules, constituting a derivation. However, not all conceivable rules are allowed, rather, they fall into fairly narrow classifications. Idsardi 1992 also required a strict ordering of the parameterized rule classes there -- for example, Edge Marking universally preceded Iterative Constituent Construction. In Idsardi and Purnell 1997 we proposed to derive such orderings from more general principles (rule complexity and the Elsewhere Condition). Although these are interesting and important questions about the nature of rule-ordering, these questions will not be addressed directly in this chapter, as the revisions to the theory generally streamline the analyses in ways that make these questions difficult to assess. In fact, Nanti (Crowhurst and Michael 2005) is an example of a language where heavy syllable marking follows binary footing.
The major changes to the theory are the following: (1) the abandonment of all Avoid constraints; clash resolution will be handled by deletion rules, as in Idaardi 1994, (2) the unification of Edge Marking rules with Iterative Constituent Construction rules into a generalized Grouping rule component, (3) the introduction of a parameter of iterativity, (4) the direct identification of parameter settings for rules with particular properties of FSAs, and (5) the projection of brackets onto higher grid lines.

With these changes we can reduce metrical calculations to two basic operations: Projection, which creates a new line of the grid from the current top line, and Grouping, which partitions the current top line of the grid into groups. Formally, this is a calculus over two types of elements: grid marks, notated with “x” and partition junctures, “(” and “)”. As with previous versions of the SBG theory, “(” groups marks to its right, and “)" groups marks to its left. A single boundary is sufficient to define a grouping, and in general the groupings do not need to exhaust the elements in the entire form. In order to project a new grid line two parameters must be specified indicating which marks and brackets should be projected, summarized in (1).

(1) Projection: Project the left/right-most element of each group

Project left/right/no brackets

The projection of “no brackets” is included for explicitness; this corresponds technically to the lack of the rule in question, not the existence of a “no” parameter value. The changes in (1) brings projection from all grid lines into accord with what was assumed to be possible in projecting syllable information onto line 0. The marking of heavy syllables (and other special syllable configurations such as Dorsey's Law vowels in Winnebago, see below) by projecting
brackets onto line 0 is an instance of (1), with a further specification of the context for syllable bracket projection. One goal of this change is to give a SBG account of syllable structure, by using brackets on the x-tier to indicate a simplified (even impoverished) syllable structure, not unlike that of Clements and Keyser 1983. I do not have space in this chapter to explicate and defend such a theory, but in general we would expect that the x-tier groups would approximate the extent of the moras in Hyman 1985, especially the first mora of a syllable in that theory, although we would also expect non-exhaustive parsing of the x-tier into syllables (i.e. there are more general cases of extrasyllabicity). One such example of non-exhaustive parsing will be the treatment of coda consonants. In particular, in some languages “coda” consonants would simply be unsyllabified, and the groupings for the syllables would then not be exhaustive. The “mora” groups will be right-headed and heavy syllable marking can project left brackets for the x-tier context x)x, as for example in English heavy syllable marking in forms like “aptitude”, (2).

(2) \( \begin{array}{ccc}
(x & x & (x \\
x)x(xx)(xx) & xx & x\text{-tier} \\
æ & p & t & I & t & u & ud & \text{features} \\
\end{array} \) line 0

To create regular groups of marks we will employ Grouping rules, using the parameter settings given in (3).

(3) Grouping: Insert left/right brackets

For every two/three elements

Starting from the left/right most element

Iteratively/Noniteratively
Starting in the \textit{insert/skip} state

This gives a total of 32 possible grouping rules. Ternary systems are not very well attested in natural language systems (though see Tripura Bangla, below), though there are poetic systems with ternary groupings (e.g. dactylic and anapestic, see Fabb 2002) and many musical examples in triple (¾) time (waltz, minuet, etc.). The 16 predicted binary patterns of grouping are given in (4), along with some example languages.

\textit{(4)}

\begin{tabular}{|c|c|c|c|c|c|c|c|}
\hline
Bracket & Size & Direction & Iterate & Start & \multicolumn{2}{|c|}{\textit{x x x x x x}} & \multicolumn{2}{|c|}{\textit{x x x x x x}} & \text{Example languages} \\
\hline
a. & L ( & 2 & L \Rightarrow & \text{Iterative} & \text{Insert} & \textit{x x(x x x x x} & \textit{x x x x x x} & \text{Maranungku, Auca stems} \\
\hline
b. & L ( & 2 & L \Rightarrow & I & \text{Skip} & \textit{x(x x x x x(x} & \textit{x x x x x x} & \text{Winnebago} \\
\hline
c. & L ( & 2 & L \Rightarrow & \text{Non-iterative} & I & \textit{(x x x x x x} & \textit{(x x x x x x} & \text{North Kyungsung Korean} \\
\hline
d. & L ( & 2 & L \Rightarrow & N & S & \textit{x(x x x x x} & \textit{x x x x x x} & \text{Tokyo Japanese} \\
\hline
e. & L ( & 2 & R \Leftarrow & I & I & \textit{x(x x x x x} & \textit{x x(x x x x} & \text{Suruwaha?} \\
\hline
f. & L ( & 2 & R \Leftarrow & I & S & \textit{x x(x x x x x} & \textit{x x x x x x} & \text{Nengone, Auca suffixes, Garawa} \\
\hline
g. & L ( & 2 & R \Leftarrow & N & I & \textit{x x x x x x} & \textit{x x x x x x} & \text{Turkish?} \\
\hline
h. & L ( & 2 & R \Leftarrow & N & S & \textit{x x x x(x x} & \textit{x x x x x x} & \text{Polish, Indonesian, Turkish?} \\
\hline
i. & R ( & 2 & L \Rightarrow & I & I & \textit{x(x x x x x} & \textit{x x x x x x} & \text{Maranungku?} \\
\hline
j. & R ( & 2 & L \Rightarrow & I & S & \textit{x x(x x x x(x} & \textit{x x x x x x x} & \text{Araucanian} \\
\hline
k. & R ( & 2 & L \Rightarrow & N & I & \textit{x x x x x x} & \textit{x x x x x x} & \text{Tauya} \\
\hline
l. & R ( & 2 & L \Rightarrow & N & S & \textit{x x x x x x} & \textit{x x(x x x x} & \text{Garawa} \\
\hline
m. & R ( & 2 & R \Leftarrow & I & I & \textit{x x(x x x x x} & \textit{x x x x x x} & \text{Suruwaha, Tauya} \\
\hline
n. & R ( & 2 & R \Leftarrow & I & S & \textit{x x x x x x} & \textit{x x x x x x} & \text{Latin, Greek} \\
\hline
o. & R ( & 2 & R \Leftarrow & N & I & \textit{x x x x x x} & \textit{x x x x x x} & \text{Russian, Japanese palatal} \\
\hline
p. & R ( & 2 & R \Leftarrow & N & S & \textit{x x x x x x} & \textit{x x x x x x} & \text{Shingazidja, Latin, Greek clitics} \\
\hline
\end{tabular}

The basic FSA for (4j) is shown in (5). By convention, the FSA starts in the state indicated by the incoming arrow.
This FSA is R2LIS, it inserts a right bracket ")" every 2 marks, from left to right, iteratively, skipping (not inserting) on the first mark. We read the arcs between states in the following way: simple arcs -- those without a colon, : -- match and consume the symbols shown on the arc (the labels are to the right of the arcs). Transductions -- arcs with a colon, : -- match the symbol to the left of the : in the input, and replace it with the symbols to the right of the :. So, in the “Skip” state, the machine accepts "x", outputs "x" and moves to the “Insert” state. In the “Insert” state, the machine accepts "x", outputs "x)" and moves back to the “Skip” state. That is, in the terminology suggested by Nigel Fabb, in moving from the “Skip” state to the “Insert” state we skip a mark, and in moving from the “Insert” state back to the “Skip” state we insert a bracket.

The double circles enclosing the state nodes indicate that the machine can stop in that state (i.e. it is legal to be at the end of the string). The present machine can stop in either state.

The full machine, which additionally respects pre-existing metrical structure (in the sense of Halle 1990), is shown in (6).
Notice that the arcs from either state when the machine encounters "(x" always go to the Insert state, and the arcs for "x)" and "(x)" always go to the skip state. That is pre-existing brackets are always treated in the same way, regardless of the current state of the machine. The embellishments to handle metrical respect are predictable from the bracket being inserted and the direction of the machine, as in the table in (7).

(7) Embellishments for respecting brackets

<table>
<thead>
<tr>
<th>Direction</th>
<th>Bracket for Insertion</th>
<th>Pre-existing Structure</th>
<th>Target State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left to right</td>
<td>Left</td>
<td>(x</td>
<td>Skip</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x) : (x</td>
<td>Insert</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(x)</td>
<td>Insert</td>
</tr>
<tr>
<td>Right</td>
<td>(x)</td>
<td>x)</td>
<td>Skip</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(x)</td>
<td>Skip</td>
</tr>
<tr>
<td>Right to left</td>
<td>Left</td>
<td>(x)</td>
<td>Skip</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x)</td>
<td>Insert</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(x)</td>
<td>Skip</td>
</tr>
<tr>
<td>Right</td>
<td>(x : (x)</td>
<td>x)</td>
<td>Insert</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(x)</td>
<td>Skip</td>
</tr>
</tbody>
</table>

7
The generalizations that emerge are that if the pre-existing bracket is the same as the one being inserted, then the machine moves to the skip state. If the opposite bracket is encountered, the machine moves to the insert state. When “(x)” is encountered, the machine moves to the insert state if the inserted bracket is the “same” as the starting position (left brackets from the left edge, right brackets from the right edge). Finally, if the inserted bracket is the same as the starting position, and the machine is in an Insert state and encounters the opposite bracket (i.e. encountering “x)” when inserting “(“ from the left edge in the Insert state or encountering “(x)” when inserting “)” from the right edge in the Insert state) then the bracket will be inserted, outputing “(x)” in both cases.

The basic FSA for R2LII, (4i), the one that inserts on the first mark instead, is shown in (8a), and the full machine is shown in (8b).

(8) a.

![Diagram](image)
These machines are identical to those for R2LIS except for the starting state. That is the incoming arrow points at the Insert state in (8), but at the Skip state in (5) and (6).

The basic FSA for L2LII, (4a), is given in (9a) and the full FSA is given in (9b).

Again, the FSA for L2LIS is the same, with the state labels reversed. That is, the machine starts
in the other state. This exhausts the binary machines, as the right to left machines simply scan the string starting from the opposite end of the string.

An example of a basic FSA for ternary parsing is shown in (10a), and the full machine is shown in (10b). These are created from the basic binary machines by fissioning the skip state. That is, the ternary machines are mechanically derivable from the binary machines.

(10) a.

Notice also that the only difference is that the basic ternary parser has two "skip" states but the same rules describe how to elaborate the basic machine to get the full machine. Thus, for iterative machines there is a basic machine, which is elaborated in the same way each time to
produce the full machine. In addition, the start state can be specified for a parsing, and the size of
the machine (number of skip states: 1 or 2) can be specified. Thus, there is a direct relation
between the parameters used here, (3), and the construction of the appropriate FSA.

The machines encountered so far have an interesting property: No look-ahead is required
with any of these machines. To understand this issue, let us consider instead a hypothetical
variant of L2LII which would require at least two marks at the end, resulting in a ternary
constituent at the far side of odd strings, a mirror image of the Garawa pattern: (xx(xx(xx, (xx(xx
(xxx. A basic machine for this is (11); the embellishments for respecting pre-existing brackets
are suppressed in this diagram.

(11)

In this machine the “Insert” state has two arcs for "x". One inserts (outputs "(x") and moves to
the “Skip” state, the other skips (outputs "x") and stays in the “Insert” state (which is obviously
not so well named in this machine). This is a non-deterministic FSA (NDFSA or NFA) because
of the choice available in the Insert state for the encountered token. But, furthermore, the “Skip”
state is not an acceptable finishing state. In that way the output "(xx(xx(xx(x" is not allowed.
However, we need to also establish the priority of inserting over skipping in non-final cases,
which could be done in NDFSAs by assigning probability values to the arcs and trying the most
probable arc before trying others. The problem with NDFSAs is that they need additional information to work properly. In order to algorithmically ensure that we don't use the skip arc from the “Insert” state when we don't need it we need to make a more complicated machine (that is, to transform the NDFSA into a deterministic FSA). Additionally, "(x" is OK at the end of the grid as long as it is pre-existing (in other words, the machine has to respect such a bracket). We can build such as machine, as shown in (12), where the arcs for “(x” have also been added.

(12)

The machine (12) is clearly more complicated than L2LII, (8), having twice as many states. Instead of making progressively more complicated elemental machines, we will instead combine elemental machines to create more complicated machines, as described in Beesley and Karttunen 2003. That is, cases like Garawa will be done by applying two footing rules in succession, as proposed in Idsardi 1992 to handle “weak local parsing” effects (Hayes 1995) in Chugach Alutiiq. In particular, Garawa-type cases will have a non-iterative footing rule, followed by an iterative one. That is, the footing that (12) would do would be done by L2RNS followed by R2LIS, giving parsings for even-numbered strings like xx)xx)(xx) and for odd-numbered strings
like xx)xx)x(xx), see also Tripura Bangla, below.

Thus, we need to address at this point what the non-iterative machines look like. As a simple case in point, we will take L2LNI, shown in (13). In the non-iterative case none of the arcs return to the Insert state. Once we are in the Skip state we will consume the rest of the string without changes; one might term this a spin state.

(13) a.

\[ \text{Insert} \xrightarrow{x:(x)} \text{Skip} \]

b.

\[ \text{Insert} \xrightarrow{(x)} \xrightarrow{x:(x)} \xrightarrow{(x)} \text{Skip} \]

The behavior of non-iterative machines that begin with a skip is not quite as clear. I give my proposal for R2LNS in (14). The question at issue is whether we should wait for a successful insertion before consuming the rest of the string, or whether the application should be vacuous under various circumstances. Given the paucity of relevant evidence at the moment, I will simply leave this question open, though (14) assumes a successful eventual insertion, which seems
Returning to the parameterized rules given in (3), we see that ternary parsing can be derived from binary parsing by fissioning a state in the binary machine. Similarly, starting in the opposite state changes the “phase” of the parsing, and directing the insertion arcs away from the starting state defines a non-iterative machine. The bracket inserted remains a fundamental choice, as does the direction in which the form is scanned. We are left with a small number of fundamental machines, none of them larger than 3 states, which cover the metrical parsings necessary to analyze counting structures in phonology.

3. **Case Studies**

In this section I will briefly consider how the revised theory would deal with certain illustrative cases that exercised the older SBG theory in several ways.

3.1. **Old English**

Idsardi 1994b offered a SBG account of the stress system of Old English. A translation into the present system is straightforward: heavy syllables begin feet and project two grid marks, the grouping is R2LIS (i.e. the first machine we considered, (5)) and each foot projects the left-most
element onto line 1. As discussed in Idsardi 1994b, there is some dispute as to whether sequences of two light syllables following a heavy syllable bore a secondary stress or not. That is, it is not clear from the available evidence whether forms such as *sealfode* consistently had a secondary stress on the second syllable. In Idsardi 1994b I suggested that this vacillation in secondary stress was due to the open foot, as shown in (15).

(15) (xx) x x)

    sealfode

One of the changes in the present theory – the projection of brackets – allows us a somewhat more principled answer to the question. If we project left brackets onto line 1, then we can make a further distinction in the stress grid, distinguishing the heads of closed feet from those of open feet, and ultimately only projecting the former onto line 2. That is, on line 0 we will project the left-most elements of each foot and project left brackets as well. Then on line 1 we will have left-headed feet (*L2LNI* is vacuous on line 1 as *L2LNI* applies on line 0 and the bracket will then project), and on line 2 we will again have *L2LNI* and left-headed feet, giving the parsing in (16).

(16) x

    (x

    (x x

    (xx) x x)

    sealfode

Thus, by building a little more structure, we can distinguish these cases by carrying forward structure from lower grid lines, in this case the left brackets from line 0 are carried onto line 1.
3.2. Auca

The stress system of Auca (Pike 1964, Hayes 1995: 182ff) is famous for its clashing stress trains.

This was not easy to handle in Idsardi 1992 because that theory assumed a tight connection between the direction of the scansion and the bracket inserted (right brackets were inserted in scansion beginning from the left-most element, and vice versa). Stems in Auca have alternating stress on odd-numbered elements counting from the left: #XxXxX...; suffixes have stress on even-numbered elements counting from the right: ...XxXx#. When there is an even number of total marks, the two stress trains are in phase and there are no conflicts and no clash. In words with an odd number of syllables, where the stress trains collide there is stress clash: XxXxX-XX, XX-xXx. This is handled by computing footing over the stem with L2LII and then re-parsing the entire form, including suffixes, with L2RIS, as shown in (17).

(17) Stem Word

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a. 5 + 2</td>
<td>(xx)(x</td>
<td>(xx)(x)-(xx</td>
</tr>
<tr>
<td>b. 2 + 3</td>
<td>(xx</td>
<td>(x-x(xx</td>
</tr>
</tbody>
</table>

The important fact is that by using left brackets in the stem parse, we leave the final foot open, and liable to “theft” of its material by the oncoming suffix parse. That is, the suffix parse can steal the last mark of the last foot of the stem to create a new foot, as in (17b). Notice that if we had used R2LIS on the stem we would have generated a lapse instead of a clash: xx)x)-(xx and xx)-x(xx.

Thus, this account generates the correct stress patterns of Auca without additional machinery beyond the parameterized values for the two scansion. In contrast, Hayes 1995 must
invoke exceptions to exceptions for Auca: Auca generally allows subminimal feet (unusual according to Hayes), but sub-minimality is revoked to allow re-parsing from the suffix train, and the resulting forms allow subminimal feet that are not located at the edge of the domain, contrary to Hayes's general principles governing sub-minimal feet. Kim 2003 analyzes Auca in OT, directly stipulating that clash occurs either across the stem-suffix boundary or at the end of the stem.

3.3. **Polish**

Polish (Rubach and Booij 1985, Idsardi 1994a), like Auca, exhibits interference between proclitic stress and stem stress. In general, the pattern of Polish is as in (18).

(18) $[\text{Clitics } \ddot{o} \rightarrow [\text{word } \ddot{o} \rightarrow \ddot{o}] \leftarrow \ddot{o}]$

Main stress occurs on the penultimate syllable of the host word and secondary stresses assigned from the left in the word and from the outside in over the clitics. The enclitics never disturb the stem stress, but it is possible for proclitic stress to disrupt secondary stresses within the word, as shown in (19).

(19)

<table>
<thead>
<tr>
<th>Word</th>
<th>None</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>\ddot{o}</td>
<td>o-\ddot{o}</td>
<td>\ddot{o}-\ddot{o}</td>
<td>\ddot{o}-\ddot{o}-\ddot{o}</td>
</tr>
<tr>
<td>3</td>
<td>o-\ddot{o}</td>
<td>\ddot{o}-\ddot{o}-\ddot{o}</td>
<td>\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>\ddot{o}-\ddot{o}</td>
<td>\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}</td>
<td>\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>\ddot{o}-\ddot{o}-\ddot{o}</td>
<td>\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}</td>
<td>\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}</td>
<td>\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}</td>
<td>\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}</td>
<td>\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}</td>
<td>\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}-\ddot{o}</td>
<td></td>
</tr>
</tbody>
</table>

Monosyllabic enclitics steal secondary stress from the initial syllable of the host word. With a
five syllable host word, the stress pattern shifts to have secondary stress on the second syllable of
the host word. Similarly, three stresses occur in the 3-3 case. But such “reparing” effects are
absent in other cases where a better binary parse would be available (1-7, 3-5, 3-7, etc.) This
limited reparing can be captured straightforwardly in the present theory by first parsing the host
word and then parsing the cliticized form. The parameterized rules are given in (20).

(20)  
a. **Stem:** L2RNS, R2LIS
b. **Clitics:** R2LNS, L2RIS
c. **Project:** L

Parsings for the various cases are shown in (21).

(21)

<table>
<thead>
<tr>
<th>Word</th>
<th>Word Feet</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>(xx)</td>
<td>(xx)</td>
<td>x-(xx)</td>
<td>xx)-(xx)</td>
<td>(xx)x-(xx)</td>
</tr>
<tr>
<td>3</td>
<td>x(xx)</td>
<td>x(xx)</td>
<td>(x-x)(xx)</td>
<td>(xx)-x(xx)</td>
<td>(xx)(x-x)(xx)</td>
</tr>
<tr>
<td>4</td>
<td>xx)(xx)</td>
<td>(xx)(xx)</td>
<td>(x-x)xx</td>
<td>(xx)-(xx)(xx)</td>
<td>(xx)x-(xx)(xx)</td>
</tr>
<tr>
<td>5</td>
<td>xx)x(xx)</td>
<td>(xx)x(xx)</td>
<td>(x-x)x(xx)</td>
<td>(xx)-(xx)x(xx)</td>
<td>(xx)x-(xx)x(xx)</td>
</tr>
<tr>
<td>7</td>
<td>xx)x(xx)</td>
<td>(xx)(xx)x(xx)</td>
<td>(x-x)x(xx)x(xx)</td>
<td>(xx)-(xx)x(xx)x(xx)</td>
<td>(xx)x-(xx)x(xx)x(xx)</td>
</tr>
</tbody>
</table>

The unary feet created with one syllable proclitics can have stress as long as they are not in clash
with a following foot. This is captured with the rule in (22); these deleted brackets are shown
struck-through in (21).

(22)  

A slightly different alternative would be to order (22) before L2RIS; then the pair of unfooted
marks in 1-5 would be picked up and footed by L2RIS. Thus, under this analysis, the “reparing”
effects in 1-5 and 3-3 forms fall out from the clitic parsing which generally respects the host word parsing.

In addition, enclitic strings will not have any right brackets in them, as L2RIS provides only left brackets. Thus, the final closed foot will be the final foot of the host word, and we can locate main stress here by projecting right brackets onto line 1, and making a right-headed constituent on line 2, as shown in the encliticized form in (23).

\[
(23) \quad x
\]
\[
x) \quad R2RNI, \text{Project: R}
\]
\[
x) \quad x \quad x \quad \text{Project: R}
\]
\[
(x \ x) \quad x \quad x \quad (x \ x) \quad \text{Project: L, )}
\]

Zabil = sie by on wam tu

Thus, the revised simplified bracketed grids provide just enough differentiation of structure to carry forward morphological information which makes it possible to locate main stress on an interior morphological constituent. In this case, right brackets are inserted on the host word cycle and never over enclitics, allowing a morphological distinction to be converted intometrical structure distinction. These effects in Polish would be difficult if not impossible to achieve with theories employing only a single juncture element (Separator Theory), as advocated by Reiss (this volume).

3.4. Winnebago

Halle and Idsardi 1995 use Winnebago as an example of a language employing several Avoid constraints. We offer here a reanalysis of Winnebago that employs only one rule of clash
resolution, substantially similar to that for Old English in Idsardi 1994. The parameter rule
settings for Winnebago are given in (24). The present analysis will eliminate the need for the
various Avoid constraints used in Halle and Idsardi 1995.

(24) a. Project (xx for CVV

    Project (x for CRV – then Dorsey’s Law gives CVRV with x(x

b. Initial Deletion: ( → ∅ / _x

    Final Deletion: ( → ∅ / _x#

c. L2LNS, R2LIS

d. Project R, )

e. R2RNI, Project R

Some example parsings are shown in (25). Dorsey’s Law vowels are capitalized.

(25) light syllables

    a. x   b. x   c. x

    x)   x)   x)

    x(x x(x x) x(x x) x

    wążê      hotaxi      hočičník

heavy syllables

d. x   e. x   f. x

    x)   x)   x x)

    x(x x(x x) x(x (x)

    wąñâk      hoočâk      nąqąwąk
The deletion rules in (24b) effectively retract some bracket projections, but the operation of Final Deletion is opaque with respect to L2LNS, as shown in (25a,d). The forms (25f,i,m) show the failure of non-final constituents open on the right to get level 2 stress, because there is no right bracket to project onto line 1. The form (25l) shows the effect of Final Deletion feeding R2LIS.

Cases with ternary parsing, that is, interior lapses of up to three syllables (Halle and Idsardi 1995: 438) occur in several contexts. Hayes (1995: 350) remarks that “[i]t appears that different morphological contexts give rise to binary or ternary patterns, in a way that is not well understood.” The ternary patterns can be generated easily in the present analysis by marking
certain additional syllables with “(x”. This can prevent footing of interior marks in an interesting way when the following syllable also has a left bracket. In (26) the syllable ga projects an extra left bracket, and since the following long-vowel syllable pui also projects a left bracket, the net effect is to prevent stress on both giš and ga, leading to the extended lapse.

\[
\begin{align*}
(26) & \quad x & x & x \\
& \quad x) & x & x) & x) \\
& \quad x(x)x & (x (xx)x x) \\
\end{align*}
\]

wayįįgišgapuižere

One way to view the net effect of the rules in (24) is to distinguish binary groups from unary ones, though note (25a,d) where unary constituents do get main stress. Because of this, Winnebago does not constitute a wholly effective argument against theories employing only a single juncture element (Separator Theory), as outlined by Reiss (this volume). In the case of Winnebago, the effect achieved by R2LIS and Project ) can also be achieved in almost all cases by restricting Projection to groups of two or more elements, and some small additional stipulation must be made for (19a,d).

3.5. **Tripura Bangla**

Beasley and Crosswhite 2003 analyze Winnebago and other languages emphasizing the use of Avoid constraints, adding Avoid #xx) to Winnebago to achieve ternary parsing at the beginning of the form: #xxx)x.... Instead, in the present proposal, as seen above, the beginning of the string is parsed as #x(xx)x.... They go on to analyze a genuinely ternary system, Tripura Bangla (Das 2002), using Avoid )xx) to force ternarity, as proposed in Idsardi 1992 for Chugach Alutiiq. The
present analysis simply admits the existence of ternary parsing systems, (10), (despite their relative rarity in stress systems). We offer here a brief re-analysis of Tripura Bangla under the present proposal. In words consisting only of light syllables stress falls on the first, fourth, seventh, etc. syllables, but not on the last syllable. Adapting Beasley and Crosswhite’s analysis to the present framework, this is equivalent to the two Grouping rules R2RNS and R3LIS with Projection of the left-most element, as shown for the abstract forms in (27).

(27)  a.  x  x  b.  x  x
     x x x) x) x  x x) x x) x

Heavy syllables generally attract stress and disrupt the count, but they group with two following light syllables showing that heavy syllables project a left boundary and only a single mark onto line 0. When there are two heavy syllables in a row, the second is not stressed, which is the effect of applying (28a) iteratively from left-to-right (the same rule is also used in Old English to resolve #LH sequences, see Idsardi 1994b, and similar rules were employed in Winnebago and Polish). The FSA for (28a) is shown in (28b).

(28)  a.  \( \rightarrow \emptyset / (x_x \)
     b.  

![Diagram](image-url)
Simultaneous application of (22a) as opposed to the iterative application observed in Tricura Bangla can be achieved by having the (x:x arc return to the “Delete” state instead of to the “Skip” state. This will remove a sequence of brackets from strings such as (x(x(x, leaving only the first, because if we stay in the “Delete” state we will keep deleting. Thus, the parameter of iterative versus simultaneous application of such rules (as discussed in Kenstowicz and Kisseberth 1977, 1979) is again directly related to an aspect of the finite state device. If the deletion is accompanied by a change in state to the Skip state, then we will get an alternating pattern of deletions. If, on the other hand, we stay in the Delete state we will get a series of deletions. Kenstowicz and Kisseberth discuss other examples of languages which differ on exactly this question with respect to rules of vowel shortening (Slovak vs. Gidabal) and jer-lowering in different Slavic dialects.

Partial derivations for HHH, LLH and LLLHH are shown in (29):

(29)

<table>
<thead>
<tr>
<th></th>
<th>a.</th>
<th>b.</th>
<th>c.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>H H H</td>
<td>L L H</td>
<td>L L L H H</td>
</tr>
<tr>
<td>Project ( for H</td>
<td>(H (H (H</td>
<td>L L (H</td>
<td>L L L (H (H</td>
</tr>
<tr>
<td>R2RNS – see (14)</td>
<td>----</td>
<td>L L)(H</td>
<td>L L L)(H (H</td>
</tr>
<tr>
<td>R3LIS</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>Clash resolution</td>
<td>(H H (H</td>
<td>----</td>
<td>L L L)(H H</td>
</tr>
<tr>
<td>Project left-most</td>
<td>x x</td>
<td>x x</td>
<td>x x</td>
</tr>
<tr>
<td></td>
<td>(H H (H</td>
<td>L L)(H</td>
<td>L L L)(H H</td>
</tr>
</tbody>
</table>

The present analysis of Tripura Bangla uses the two available brackets to encode different information in feet – left brackets carry forward heavy syllable information. In separator theory
where there is only one grouping juncture \( \downarrow \), this cannot be done. This raises
problems in Tripura Bangla in distinguishing between final L, which does not bear stress, and
final H which does. Obviously, an additional rule could be added stipulating that final H projects
\( \downarrow \downarrow \), whereas non-final H projects only \( \downarrow \). This is clearly more complicated than projecting \( \downarrow \) for H uniformly, and so we have a trade-off between the complexity of additional representational
possibilities and the complexity of additional rules. Likewise, the separator theory without
further adornments will make different predictions for clash resolution for strings such as those
in (30) (it should be noted that cases like (30b) are not as yet attested):

(30)    SBG    a.  (H (H L) L    b.  L L L) L (H L) L

Separator  a.  \( \downarrow H \downarrow H \downarrow L \downarrow L \)  b.  L L L \( \downarrow L \downarrow H \downarrow L \downarrow L \)

Two responses are possible in Separator theory, (1) to state a second clash resolution rule for the
\( \downarrow L \downarrow H \) sequences, or (2) to apply clash resolution before any grouping operations. Since rules must
be ordered anyway, ordering clash resolution would seem to pose no particular problems, though
the ordering of Clash resolution would be later in Old English. Thus, although Tripura Bangla
does not provide any definitive argument against Separator theory, it does illustrate the tradeoff
between additional representational abilities and the necessity of additional rule computations in
the absence of representational possibilities.

4. Automata for Optimality Theory

Karttunen 1998 demonstrates how to construct FSAs from OT grammars using the lenient
composition operator \( .O. \) in the Xerox finite-state calculus. This operator is available (but not
documented) in Beesley and Karttunen 2003, so the main source on the use of lenient
composition remains Karttunen 1998. As an exercise we will construct a FSA using OT and lenient composition to compute R2LIS, (4j), the machine that we started our discussion with. Recall that the basic SBG machine is shown in (5), and the full machine is shown in (6).

The OT equivalent for R2LIS without regard for headedness was constructed using constraints of the form advocated by McCarthy 2003. The constraints used, in their ranking order, is given in (31).

(31)  a. FtBinMax: No more than two elements per foot
       b. FtBinMin: At least two elements per foot
       c. Parse2: No pairs of unparsed elements
       d. Parse: No unparsed elements
       e. Unparsed-At-End: No initial or medial unparsed elements (compare Lapse-At-End)

This constraint ranking produces parses as in (32).

(32)  a. Even: (xx)(xx)(xx),
       b. Odd: (xx)(xx)(xx)x

Leniently composing these constraints using Karttunen 1998 produces the FSA in (33). This machine has 41 states and 123 arcs, compared to the 2 states and 8 arcs of the full machine in (4).
This major difference in the size of the automata raises many questions, some of which cannot be answered at this point. Does the software work properly? That is, is the FSA properly reduced to minimum size? Is lenient composition the wrong way to construct FSAs from OT constraints? Is there a problem with the constraints in (31)? Do certain combinations of constraints cause the explosion in machine size? Most of these questions require additional investigation. One point
made by Karttunen 1998 is clear, however. The use of constraints with an unbounded number of violations possible is a major source of computational complexity.

(34) “It is immediately evident that while we can construct a cascade of constraints that prefer \( n \) violations to \( n+1 \) violations up to any given \( n \), there is no way in a finite-state system to express the general idea that fewer violations is better than more violations.” Karttunen 1998: 11.

But an unbounded number of violations is exactly what McCarthy's 2003 proposal requires. Although it eliminates the gradient evaluation of each violation instance, a given form can have unboundedly many instances of a violation-type, as the quote in (35) shows.

(35) “… it is sufficient for any constraint to assign one violation-mark for each instance of the marked structure or unfaithful mapping in the candidate under evaluation.” McCarthy 2003: 130 (emphasis added)

In the present example all the constraints in (31) can have unboundedly many violations in a given candidate. Karttunen goes on to explore the implications of this finding, quoted in (36).

(36) “It is curious that violation counting should emerge as the crucial issue that potentially pushes optimality theory out of the finite-state domain thus making it formally more powerful than rewrite systems and two-level models.” Karttunen 1998:11

Combining Karttunen's findings with our own, we conclude that OT is formally more powerful than rule-based theories, and is also orders of magnitude less efficient in calculating simple parsing problems such as (4j) which can be done with the minimal possible machine in rule-based theory.
5. Conclusions

In this chapter I have offered some revisions of SBG theory, eliminating Avoid constraints, combining the mechanisms of Edge Marking and Iterative Constituent Construction into a general footing system and relating the new footing system to FSAs. I have shown that the resulting FSAs are small and efficient, orders of magnitude smaller than the corresponding OT machines complied according to Karttunen 1998. The revisions of the theory allowed for simpler accounts of difficult cases such as Old English, Auca, Polish, Winnebago and Tripura Bangla, representing a significant improvement in the theory. These cases also highlight the utility of the distinction between open and closed feet, and demonstrate the utility and efficiency of employing distinct grouping junctures “(“ and “)”.

References


Fabb, N. 2002. Language and literary structure: The linguistic analysis of form in verse and
narrative. Cambridge: Cambridge University Press.


30


