Let the decimal system do it for you

a very simple utility function for OT

Alan Prince
December 16, 2007

Abstract

A pen-and-paper procedure turns OT violation profiles into numerical values that faithfully reproduce the harmonic order on candidates, no matter how many candidates there are, no matter how many violations or how many constraints. The method belongs to the family of utility functions introduced in Prince 2006a, providing a huge simplification of the techniques discussed there. Some general properties are noted which distinguish an actual OT utility function from weighted-sum approximations.

Thanks to Bruce Tesar, Ed Keer and Vieri Samek-Lodovici for useful discussion.

0. Background

OT famously cannot be constructed as numerical weighting of constraints (Prince & Smolensky 2004:236; Prince 2002:1, 2006a:1; Legendre et al. 2006:342ff). A minimal example tells the story:

(1) $\mathbf{C_1 \gg C_2}$

	C_1	C_2
☞ **a**	0	2
b	1	0

We want to assign a numerical score to each candidate, based on its violation pattern, such that the OT winner **a** has a *smaller* score than **b**. Suppose we speculatively declare C_1 to be twice as weighty as C_2 and calculate as follows. (We write $|\mathbf{x}|$ for the score associated with **x**, w_k for the weight of C_k).

(2) **The Reckoning:** $w_1 = 2$, $w_2 = 1$
- i) $|\mathbf{a}| = \mathbf{2}\cdot 0 + \mathbf{1}\cdot 2 = \mathbf{2}$
- ii) $|\mathbf{b}| = \mathbf{2}\cdot 1 + \mathbf{1}\cdot 0 = \mathbf{2}$

(Equivalent results will come from setting the first weight at 1 and the second at ½.)

But we haven't succeeded: the competitors come out the same. A way out is clear: raise the weight of the dominating constraint. In this particular case, any increase at all will tilt the scales in the right direction (we continue with integer weights, for simplicity).

(3) **The Reckoning II:** $w_1 = 3$, $w_2 = 1$
- i) $|\mathbf{a}| = \mathbf{3}\cdot 0 + \mathbf{1}\cdot 2 = \mathbf{2}$ ☜
- ii) $|\mathbf{b}| = \mathbf{3}\cdot 1 + \mathbf{1}\cdot 0 = \mathbf{3}$

The OT winner now earns the minimal score (2<3), as desired. (Success would also be achieved by setting the first weight to 1 and the second to ⅓.) But this example already contains the seeds of the method's downfall. Greater violation disparity in C_2 can easily push the weighting scheme into anti-OT behavior.

(4) **Against the numerical grain**

	C_1	C_2
☞ **a′**	0	**3**
b	1	0

Using the same scheme, we get |**a′**| = |**b**|. And should we encounter some **a″** with *four* violations of C_2, the OT preference will be inverted. Bruce Tesar remarks that in any such pattern, when the OT winner's C_2 violations equal or exceed the value of the ratio w_1/w_2, the weighting will fail to deliver the OT outcome (Tesar 2007).

The moral is this: any increase in the relative weight of C_1 can be sabotaged. All that's needed is an appropriately dismal performance on C_2 by the OT winner. No matter how we weight the constraints, the scheme can always be overcome. There's always a perverse configuration in which the OT winner barely squeaks by on the *dominant* constraint and does massively worse than its competitor on the subordinate one, with the sheer number of low-ranked violations collapsing the weight scheme.

This observation generalizes to any number of constraints.[1] To get a sense of how things go, let's look at a three-constraint system, with a couple of new candidate sets.

(5) **A longer run**: $C_1 \gg C_2 \gg C_3$

		C_1	C_2	C_3
cset 1	☞ **a**	0	0	**2**
	b	0	1	0
cset 2	☞ **c**	0	**2**	**2**
	d	1	0	0

The *first* contest, **a** *vs*. **b**, exactly mirrors our simplest 2-constraint example. It can therefore be correctly represented by weighing C_2 in at 3× the value of C_3. The second candidate set is more interesting, since the OT winner **c** is encumbered by violations of both subordinate constraints. Nevertheless, the problem is fundamentally the same, and the same solution will work: $\mathbf{C_1}$ can be successfully weighted at 3× the weight of C_2. This constraint already bears 3× the weight of the base case C_3, so the C_1 weight is $3 \cdot 3 = 9\times$ the weight of bottommost C_3.

(6) **The longer calculation:** $w_1=9$, $w_2=3$, $w_3 = 1$

i) $|\mathbf{c}| = \mathbf{3 \cdot 3} \cdot 0 + \mathbf{3} \cdot 2 + \mathbf{1} \cdot 2 = \mathbf{8}$ ☜

ii) $|\mathbf{d}| = \mathbf{3 \cdot 3} \cdot 1 + \mathbf{3} \cdot 0 + \mathbf{1} \cdot 0 = \mathbf{9}$

[1] In any finite collection of data, there must be a maximal violation value on each constraint; therefore, an OT grammar's action on any such collection can be modeled by a weighting scheme. This is a long way from deriving the linguistic generalization, which is about the language, not a sample of it.

This progression signals the exponential increase in weights that will be required in the general case. Here we see the emergence of the pattern 3^n , …,3^3, 3^2, 3, 1 that will be appropriate for n constraints where the largest violation quantity is 2. (An equivalent weight sequence is 1, ⅓, $^1/_9$, $^1/_{27}$, …, $^1/_{3^n}$ — it's all about the ratios.)

The shadow of a positional representation comes near. Suppose we simply run together the individual numerals in the *violation profile*, dropping an initial run of zeros for temporary convenience, and write ‖**x**‖ for the result.

(7) **Positional numeration**

value	violation sequence
i) ‖**a**‖ = 2	0⌢0⌢2
ii) ‖**b**‖ = 10	0⌢1⌢0
iii) ‖**c**‖ = 22	0⌢2⌢2
iv) ‖**d**‖ = 100	1⌢0⌢0

If these are read in the normal way, we get exactly the less-than/greater-than relations that we need:

desired: **a** ≻**b**	obtained: ‖**a**‖ < ‖**b**‖	2 < 10
desired: **c** ≻**d**	obtained: ‖**c**‖ < ‖**d**‖	22 < 100

We can read the numerals of ex. (7) as base-3, as decimal, or indeed as any base whatever that admits digits 0,1,2. In each case, we recover the candidate order imposed by ranking.

The reason this works is because positional numerals are based on exponential weighting of digits and ordered by the same principle as OT candidates. For any two such numeric expressions, their less-than/greater-than relationship is fully determined by *the first digit on which they differ* (scanning left to right, and padding with initial zeros to make them the same length). This is 'lexicographic' or 'dictionary' order in the parlance of the topologists (Hausdorff 1908), so called because *words* on an alphabet — strings of letters just as numerals are strings of digits — are so ordered by the underlying alphabet's order.

No amount of failure on lower-ranked constraints can bring down the OT winner; similarly, no finite amount of excess on the rightward digits can upset an established numerical order. The expression .999…9 is less than 1, regardless of its loading with 9's.[2]

The decimal system, like any other positional scheme, is locked into a fixed rate of change (albeit exponential) between positions. And like any such scheme, OT can outrun it. Our first example-type, as promised, is sufficient to tell the story :

(8) $\mathbf{C_1 \gg C_2}$

	C_1	C_2
☞ **a‴**	0	10
b	1	0

[2] Even infinite end-loading with digits other than 9 will not succeed. While .9999…*ad-infinitum* is the same as 1 in the decimal system, the infinite sequence .333… is uniquely 1/3.

This defies decimal representation because violation of C_2 leaps beyond the base-10 digitary, and correspondingly, the example cannot be faithfully rendered by 10-based exponential increase. And any base will succumb to the same tactic.

But all is not lost: there are more ways than one to string a digit.

1. An astonishingly simple utility function for OT

Let's begin with the simplest base of all: 1. Translate violation numbers into the apositional *tally* representation, in which one is 1, two is 11, three is 111, … , seven is 1111111, and so on. We supply one 1 for each violation mark — in essence, this is exactly the original method of listing violations, with asterisks turned to tally strokes, each read as 'one'.

Now take the tally from C_1, append a 0; attach after that the tally from C_2, append a zero; attach the tally from C_3, append a 0, and so on through the domination order. (We could safely leave off the zero after the last constraint, but we'll keep it for purposes of simplicity.)

To see how this works out, let's return to our original example:

(9) $\mathbf{C_1 \gg C_2}$

	C_1	C_2
☞ **a**	0	2
b	1	0

Converting (back) to the tally:

(10) $\mathbf{C_1 \gg C_2}$

	C_1	C_2
☞ **a**		11
b	1	

The described method runs like this:

a: C_1 has no tally. Append a '0' to this nothing. Result: **0**.
C_2 tallies two violations with '11'.Append '0'; append this **110** to the C_1 result.
a → 0⌒110 = **0110**

b: C_1 tallies '1'. Append a zero. Result: **10**
C_2 has no tally. Append a '**0**' and quit
b → 10⌒0 = **100**

To use these numerals effectively, we need to impose the correct sense of exponential change between positions: blow-up (right to left), or equivalently, decay (left to right). We go for decay.

Two moves are required.

First, drop the tally reading and interpret the resulting numeral according to some positional system, any one will do (*e.g.*, decimal is fine).

Second, to ensure that weighting runs from heavy to light, respecting constraint order, interpret the expression as *fractional* by appending a period at the beginning.

Applying this recipe to the case at hand, writing ||**x**|| for result with **x**:

(11) **Positional numeration II**

	value	violations	structure
i)	||**a**|| = **.0110**	0, 2	•⌒ 0⌒ 110
ii)	||**b**|| = **.100**	1, 0	•⌒ 10⌒ 0

Observe that ||**a**|| < ||**b**||, as desired.

The same procedure extends seamlessly to *any* case, e.g. our own example(5):

(12) **The Longer Run**: $C_1 \gg C_2 \gg C_3$

		C_1	C_2	C_3
cset 1	☞ **a**	0	0	**2**
	b	0	1	0
cset 2	☞ **c**	0	**2**	**2**
	d	1	0	0

(13) **Positional numeration III**

	value	viols.	structure
i)	||**a**|| = .00110	0, 0, 2	•⌒ 0⌒ 0⌒ 110
ii)	||**b**|| = .0100	0, 1, 0	•⌒ 0⌒ 10⌒ 0
iii)	||**c**|| = .0110110	0, 2, 2	•⌒ 0⌒ 110⌒ 110
iv)	||**d**|| = **.**1000	1, 0, 0	•⌒ 10⌒ 0⌒ 0

The first constraint's contribution is demarcated by the first 0, the second's by the second, and so on. (The last constraint's tally ends the expression and is easy to find without the trailing 0, but there's no harm in keeping it.)

Here again, the assigned numerical values keep perfect track of the OT candidate ordering. The fact that ||**c**|| takes the form **.0**⌒*stuff* while ||**d**|| reads **.1**⌒*other-stuff* is sufficient to ensure that **c** betters **d**. Victory in the first constraint translates to victory in its numerical representation, no matter what follows.

In sum: take any violation profile — say (0,3,1,2) — convert the numbers to tallies and attach a final zero to each non-zero entry — yielding here (0, 1110, 10, 110) — run the whole thing together, and prefix with a decimal point — yielding .0111010110. That's it.

We are done. The numbers we obtain will rewrite the violation profiles as quantities whose order is exactly the OT order. We have created the desired harmony or 'utility' function, which faithfully translates a preference order into a numerical order. (We invert the economists' sense, though: they wish to maximize utility, where we wish to minimize our value, since it measures badness, not benefit.) All that is required is that a constraint assess a finite number of violations for each candidate.[3]

To see why this works, consider the comparison of two candidates, call them **x** and **y**. Suppose they differ on the very first constraint. If **x** satisfies the constraint and **y** does not, then $\|\mathbf{x}\| = .0\ldots$ as against $\|\mathbf{y}\| = .1\ldots$. Clearly, just as in the **c** vs. **d** case above, the numerical value of $\|\mathbf{x}\|$ can never equal exceed that of $\|\mathbf{y}\|$, no matter what happens in the rightward regions of either one.

The more interesting case arises when *both* violate the first constraint to some degree. Suppose that **x** has k violations and **y** has more than that. Let's look at the first k+1 digits of their representations:

$\|\mathbf{x}\| = .1\text{---}10\ldots$
$\|\mathbf{y}\| = .1\text{---}11\ldots$

Here again, it should be clear that $\|\mathbf{x}\|$ is irredeemably less than $\|\mathbf{y}\|$.

To get the complete picture, we need to consider two candidates whose relation is decided by some constraint in a general position in the hierarchy, say the p^{th}. The competitors must therefore *agree* on the first $p-1$ constraints. Let's examine their representations at the point where the $(p-1)^{st}$ instance of '0' appears. Let m and n symbolize the digit strings occurring immediately after the relevant '0'.

$\|\mathbf{x}\| =$.--- **0** m
$\|\mathbf{y}\| =$.--- **0** n

We are, in effect, back to the first case, since we are really only comparing m and n.

To work this out in more detail, note that the initial sequences symbolized by '.---0' must be identical, since identical violation profiles yield identical numerals. Because the internal content of the initial sequence '.---0 ' contributes nothing at all to the decision, we can replace the entire sequence with an equal length of 0's, with no effect on the relation of $\|\mathbf{x}\|$ and $\|\mathbf{y}\|$. Call the results of this replacement $\|\mathbf{x}'\|$ and $\|\mathbf{y}'\|$. The sequence '.---0' has a certain determinate length, call it k. Now multiply $\|\mathbf{x}'\|$ and $\|\mathbf{y}'\|$by 10^k. This has no effect on their relative magnitudes, either. But it reduces them exactly to the case argued above, where the candidates differ on the very first constraint.

We close by noting that the choice of 1 as the tally digit is arbitrary: any other will do as well. We don't even have to restrict ourselves to one particular digit: we're safe if we use the same digit to tally the violations of a given constraint. If we want a more variegated representation of ex. (13), we could (among many other choices) have the tally digit reflect the position of the constraint in the hierarchy.

---

[3] Were a constraint able to sponsor an infinite sequence of 1's, we'd never get to the next constraint in the hierarchy. It is, however, not required that the *number* of constraints be finite! We have a faithful numerical representation for the lexicographic order on N^k, including — remarkably — N^ω.

(14) **Variety in positional expression**

value	viols	structure
i) $\|\|\mathbf{a}\|\| = .00330$	0, 0, 2	•⁀ 0⁀ 0⁀ 330
ii) $\|\|\mathbf{b}\|\| = .0200$	0, 1, 0	•⁀ 0⁀ 20⁀ 0
iii) $\|\|\mathbf{c}\|\| = .0220330$	0, 2, 2	•⁀ 0⁀ 220⁀ 330
iv) $\|\|\mathbf{d}\|\| = .1000$	1, 0, 0	•⁀ 10⁀ 0⁀ 0

2. The Distances

Wherein lies the key difference between the utility function described here and the notion of a weighted sum of violations, the famous failure? The original discussion in Prince & Smolensky 1993/2004:236 is correct in all essentials, but the advent of concrete utility functions for OT makes it possible to see a little further into the issues.

This question assumes more than formal force because of continuing interest in a species of Harmonic Grammar (Legendre, Miyata, & Smolensky 1990abc) limited to positive weights (Pater, Potts, & Bhatt 2006; Pater, Bhatt, & Potts 2007, Tesar 2007, and references therein).

2.1 How far from weights?

In the faithful quantitative map of OT developed here, as well as in any partially faithful weighted-sum approach, constraints are weighted exponentially. The 0 appended to the tally has the effect of jogging the value of the next constraint's violation downward by an extra 1/10 (thinking *decimal* system), thereby distinguishing it from the preceding constraint. Crucially, though, this weighting is dynamic and depends not just on the position of the constraint in the hierarchy, but on the actual number of violations received by the candidate on all higher-ranked constraints (*v*. Prince 2006a for the general case).

Secondly, and also crucially, the *violations* within a single constraint are also weighted, indeed exponentially discounted. One violation of the topmost constraint is worth $^1/_{10}$ and two violations are worth $^1/_{10} + ^1/_{100}$, three are worth $^1/_{10} + ^1/_{100} + ^1/_{1000}$, and so on. This means that the value of any constraint's violations is capped. For example, the first constraint's contribution never reaches $^1/_9 = .11111\ldots$*ad-inf*.

To see how far this result is from the logic of weighted sums, consider the fact that a positively-weighted sum of non-negative integers provides us with something that is conceptually close to *length*.

Think of the violation profile as a *vector* with one coordinate for each constraint. The first picture one encounters of a vector is of an arrow with length and direction. Place this in a coordinate environment, and the arrow transmutes to a list of numbers.[4] From this list one may calculate length and direction. What, then, is *length*?

[4] The list idea gives a far more general representation of the notion 'vector', displacing the 'arrow'.

Euclidean length, hooked into the Pythagorean Theorem, is the familiar idea — the square root of the sum of the squares of the coordinates. Looking a bit more deeply, analysts have extracted the essence of the length concept, under the rubric of 'norm', and found many other recipes that satisfy the basic properties.[5] Among these is the so-called 1-norm: the sum of the absolute values of the coordinates.[6] In the case at hand, all coordinates are nonnegative, and their unadorned sum gives the 1-norm *tout simple*.[7]

As it happens, weighting the coordinates with positive quantities still leaves you with a norm.[8] To check this, note that the defining properties of the norm are these:

(15) **The Norm**

[1] it is always nonnegative, and zero only when all coordinates are zero,

[2] multiplying all coordinates by the same positive value also multiplies the norm by that value, and

[3] it obeys the 'triangle inequality', which holds that the norm of the sum of two vectors is less than or equal to the sum of their norms.[9]

The margins of this paper should provide sufficient room for the reader to calculate that the positively weighted sum of a violation vector's entries does indeed satisfy these criteria.

With length in hand, a notion of *distance* between two vectors quickly follows: this is the length of the difference between two vectors, obtained by subtracting one from the other, coordinate by coordinate.

The utility function described here is emphatically *not* a norm. It doesn't give a notion of 'length' in anything like the usual sense, and therefore makes *distance* undefinable. Although the first property ('positive definiteness') is satisfied, the second ('positive homogeneity') fails dramatically. Doubling the number of violations, e.g., *never* doubles their utility value. The triangle inequality is perhaps more promising, but adding two utilities is not even guaranteed to yield a well-formed utility (e.g. 1+1=2, which isn't in the vocabulary).

Stepping back from these dire considerations, one might ask whether there is any general rapprochement available between weighting and lexicographic order of the OT type. A basic, hopeful observation is that any *finite* collection of data can be modeled by a weight system, which indeed can show weight-growth patterns far more moderate than exponential (Prince 2002; Pater, Bhatt, & Potts 2007, Pater, Potts, & Bhatt 2006).

---

[5] The Wikipedia article "Norm_(mathematics)" provides an accessible summary.

[6] This is also known (optimistically) as the taxicab norm, because it gives the minimal distance you have to travel to get from one point to another in a square grid. For example, to travel from the origin (0,0) to the point (3,2) you have to go 5 blocks — you must go at least 3 blocks east and 2 north on any path.

[7] The 1-norm is already known to play a role in linguistic theory. See Prince 2002, "Anything goes."

[8] One which is appropriate to a grid with rectangular rather than simply square blocks (*mutatis mutandis* in higher dimensions).

[9] Recall that in the arrow picture, the sum of two vectors is the third side that closes the triangle obtained by laying the summing vectors head-to-tail. Hence the name, because the length of one side of triangle is less than or equal to the sum of the lengths of the other two sides. To connect to the definition give in the text, recall that vectors *qua* lists are summed coordinate by coordinate; this gives meaning to the notion 'sum of two vectors'. (The norm is just a number, so we already know what 'sum' means there.)

To this, the immediate reply is that there are surely an unbounded number of candidate sets, and within each candidate set an unbounded number of candidates — given minimal assumptions shared by a broad spectrum of linguistic theories. Arbitrarily declaring that only *n*-many candidates will be considered, for some *n* picked on grounds of convenience, does not carry the clarion ring of principled explanation.

An OT grammar, though it deals with unbounded data, is nevertheless finitely determined. There's only a finite number of rankings on a finite constraint set. Typically, the same *language* (construed extensionally as an exhaustive collection of optima) may be generated by several such rankings, no threat to finiteness.

To connect with the data that gives rise to the grammars, observe that a language is delimited by a set of ERCs,[10] which collectively define the rankings that generate the language. An ERC emerges from the comparison between a desired optimum and a single competitor — a mere two candidates. In short: not only is there a finite number of languages in total; there exists for any given language a finite set of candidates that fully determines it: their fate determines the fate of the infinitude of others.[11] Adopting the term 'Support' (Tesar & Prince 2003) for the set of candidates giving rise to an ERC set, we see that any language has a finite Support under OT.

Call a Support 'complete' if it generates an ERC set that yields the entirety of ranking conditions needed to give a language. A complete Support is just a finite set of candidates; therefore, any such can be generated by a weighted sum of violation profiles. From this it does *not* follow that the entire language can be generated by that set of weights. But a complete Support provides a reasonable place to start looking for effects derivable from the formal properties of the weighted sum, such as normhood, which might generalize throughout the full system. If anything like this proves feasible, it might be possible to use weighting ideas (albeit carefully) to illuminate properties of full-blown grammar by looking at restricted images of it.[12]

In the end, as at the beginning, OT and weighted-sum theories remain fundamentally distinct, despite regions of descriptive overlap. Here's a couple of examples that give a concrete sense of the formal conclusions we've just reviewed.

As a first simple instance of the divergence of the two ideas, consider the absolutely basic fact that nothing prevents all weights from being set to the same value. This would mean that all candidates sharing the same *total* number of violations would be accounted equivalent, and that all would be co-optimal when that value was minimal. This has no correlate in OT, and is deeply contrary to the OT sense that violations of different constraints do not trade off. (Weighting in general will allow much richer trade-offs.)

---

[10] ERC = elementary ranking condition. Prince 2006b gives a recent introductory overview.

[11] This accords with and justifies the descriptive practice of scanning a small collection of candidates for the ERCs they require; though not, of course, with the practice of merely assuming that the candidates you happen to have scanned actually determine the language (i.e. the full set of rankings needed to decide all optima). See Prince & Smolensky 1993/2004:ch. 7 for techniques relevant to the latter problem. Prince 2006b:52ff discusses the relation between ERC sets and grammars.

[12] Analogy: we can use the formula derived for the sum of geometric progression involving a strictly finite number of terms to derive the outcome, if there is one, when the number of terms goes to infinity. This does not mean that the theory of finite series is a sub-theory of the theory of infinite series, or vice versa.

Second, and more subtly, certain aspects of harmonic bounding will disappear. Simple harmonic bounding is a widely-found phenomenon in many kinds of optimization, as one may intuit from the fact that the bounded form has no constraint that strictly favors it over its bound, i.e. has nothing in particular going for it. (Another hint of its generality is that simple harmonic bounding is co-extensive with Pareto sub-optimality.[13]) The bounding of a form by a gang of others — the 'collective bounding' of Samek-Lodovici & Prince 1999 — is much more restricted in its appearance in optimization systems.

(16) **A Painful Case**

	C_1	C_2
a	0	**3**
b	1	1
c	**3**	0

Candidate **b** cannot win under OT; it is blocked out on one ranking by **a** and on the other by **c**. It is *collectively* bounded. Nevertheless, candidate **b** wins hands down over both of its competitors when the weights are equal, and indeed whenever the larger weight is less than twice the smaller.[14] These cases indicate that the differences between weighted sum theories and OT appear not in abstruse, craftily constructed high-violation contexts, but in the most basic matters that are going to drastically influence predicted typologies.

Collective bounding, though it may appear on first exposure to be a subtlety suitable only for contemplation by initiates, is in fact closely connected with the finiteness (and indeed limited size) of OT typologies. The number of potential forms that can be *un*bounded in the simple (Pareto) sense of the term is unlimited. But the number of potential OT optima — profiles unbounded in any sense — is always limited by the size of the constraint set, and even in the most generous and symmetric of circumstances cannot exceed the factorial of the number of constraints.

To see what's going on, consider any set of forms whose violation profiles sum to the same value, call it *k*. None of these can simply bound any of the others.[15] (If **a** simply bounds **b**, the sum of coordinates of **a** must be strictly less than the sum of the coordinates of **b**, because every coordinate of **a** is less than or equal to the corresponding coordinate of **b**.) The number of profiles that sum to *k* obviously grows unboundedly with *k* — in fact it grows very rapidly. If *n* is the number of constraints, the number of profiles summing to *k* is *n+k–1 choose k*. (See appendix.) Suppose, for example, there are 5 constraints; the number of profiles whose violations sum to 10 is 1001. Yet there are only 5! = 120 possible optima, at the absolute maximum, because that's the number of different constraint rankings.

[13] An informative introduction to this concept is just a click away: see "Pareto efficiency" in Wikipedia.
[14] Let the weights be m and n . Then for **b** to win, we must have m+n<3n, *i.e.* m<2n. Symmetrically, n<2m. Suppose without loss of generality that m≤n. Then m≤n<2m.
[15] This observation comes from joint work with Vieri Samek-Lodovici, as does the idea of studying the set of profiles with the same violation sum

2.2 Weighed in the balance

Significant advances in the analysis of the relation between OT-style lexicographic optimization and Harmonic Grammar-style weighted-sum approaches are found in Tesar 2007. Interesting lines of development of the weighted-sum idea are proposed in Pater, Bhatt, and Potts 2006, 2007, but they do not characterize the OT/weighted-sum relation with sufficient exactitude to support their comparisons with OT. Prince 2002 identifies the situations in which any weighting that respects the ranking order (higher ranked = greater weight) will yield the OT optima. The key notion of harmonic bounding is explored in detail in Samek-Lodovici & Prince 1999, 2005.

3. Proof positive

Above we provided a motivating sketch of the way the positional utility function works. If we want solid proof that it is lexicographically faithful, we need look no further than "Harmony at Base Omega" (HBO: Prince 2006a), which introduces a general class of OT utility functions and proves their correctness. The relevant observation is simply that the function described here falls into that class.

A utility function in the HBO mold is built from a strictly increasing function from the nonnegative integers into [0,1). The transmutation of integers from tally to decimal fraction performs this strictly-increasing mapping into [0,$^1/_9$), which fills the bill:

$$n \mapsto \sum_{k=1}^{n} 10^{-k}$$

This maps 1 to .1; 2 to .1+.01 = .11; and so on. Given the assumption that the sum symbol yields 0 when it runs backwards from k=1 to 0, it maps 0 to 0, as desired.

The value so obtained, when non-initial, is weighted by difference between the immediately previous constraint's actual violation count and that count +1:

$$\Delta = \sum_{1}^{n+1} 10^{-k} - \sum_{1}^{n} 10^{-k} = 10^{-(n+1)}$$

This yields exactly the zero that's dropped in to demarcate the end of a constraint's contributed value. For example, if the first constraint shows 3 violations, we get .111 for its contribution. If the second constraints shows 1 violation, then its basic value is .1, and we add this in to the sum weighted by the appropriate Δ based on the 3 violations of the immediately preceding constraint, so that $\Delta=10^{-4}$. Whence

$.111 + 10^{-4}(.1) = .111 + (.00001) = .11101$

To fill in the details: HBO recursively defines a utility function, F: $\mathbb{N}^n \rightarrow [0,1)$, as follows. Notational conventions:

- $\mathbf{v} \in \mathbb{N}^n$
- $v_k = \mathbf{v}[k]$, the k^{th} coordinate of $\mathbf{v}$
- $\mathbf{v}\backslash 1$ = the list $\mathbf{v}$ with the first coordinate removed.
- $\boldsymbol{\lambda}$ the empty list

(17) **Utility Functions for OT**

Let f: $\mathbb{N} \rightarrow [0,1)$ be strictly increasing.
Let $\Delta_k = f(v_k+1) - f(v_k)$
$F(\boldsymbol{\lambda}) = 0$
$F(\mathbf{v}) = f(v_1) + \Delta_1 \cdot F(\mathbf{v}\backslash 1)$

Any utility function that satisfies these conditions faithfully renders the lexicographic order into the usual order on $\mathbb{R}$, the real numbers, as shown in HBO:5ff.

Appendix: Counting a class of non-simply bounded profiles [16]

Assuming *n* constraints, consider the collection of all violation profiles $\mathbf{v} \in \mathbb{N}^n$ meeting the following condition, for some fixed positive *k*.

$$\sum_{j=1}^{n} \mathbf{v}[j] = k$$

We need to count the number of solutions to this equation; that is, how many ways we may assign non-negative integers to the *n* variables **v**[j] so that they sum to *k*. This question is a fixture of basic probability theory, equivalent to asking how many ways *k* balls may be distributed among *n* urns; in this case, *k* violations among *n* constraints. Feller 1950:36 provides the answer, which he notates $A_{k,n}$:

$$A_{k,n} = \binom{n+k-1}{k}$$

[16] Thanks to Bruce Tesar for a simple combinatoric argument that led me away from geometry.

References

ROA= http://roa.rutgers.edu

Feller, W. 1950. *An Introduction to Probability Theory and its Applications*. John Wiley & Sons, Inc. New York.

Hausdorff, F. 1908. Grundzüge einer Theorie der geordneten Mengen, *Mathematische Annalen*, 65:435-505.

Legendre, G., Y. Miyata, & P. Smolensky.1990a. Can connectionism contribute to syntax? Harmonic Grammar, with an application. *Proceedings of the 26th Meeting of the Chicago Linguistic Society*. Chicago, IL.

Legendre, G., Y. Miyata, & P. Smolensky.1990b. Harmonic Grammar — A formal multi-level connectionist theory of linguistic well-formedness: An application. *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society*, Cambridge, MA. p. 884-891.

Legendre, G., Y. Miyata, & P. Smolensky. 1990c. Harmonic Grammar — A formal multi-level connectionist theory of linguistic well-formedness: Theoretical foundations. *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society*, Cambridge, MA. p. 388-395.

Legendre, G., Sorace, A., and Smolensky, P. 2006. The Optimality Theory–Harmonic Grammar connection. In P. Smolensky and G. Legendre 2006, vol. II, 339–402.

McCarthy, J. 2003. *Optimality Theory in Phonology: A Reader*. Malden, MA and Oxford: Blackwell.

Moreton, Elliott. 2003. Non-computable functions in Optimality Theory. In McCarthy 2003, 141-164. Also ROA-364 (1999).

Pater, J., R. Bhatt, & C. Potts. 2007. Linguistic optimization. ROA-924.

Pater, J., C. Potts, & R. Bhatt.2006. Harmonic Grammar with Linear Programming. ROA-872.

Prince, A. and P. Smolensky. 1993/2004. Optimality Theory: *Constraint Interaction in Generative Grammar*. ROA-537 & Blackwell.

Prince, A. 2002. Anything goes. ROA-536 & *A New Century of Phonology and Phonological Theory*,T. Honma, M. Okazaki, T. Tabata, & S. Tanaka. Kaitakusha: Tokyo, pp. 66-90.

Prince, A. 2006a. Harmony at base omega. ROA-798.

Prince, A. 2006b. Implication & Impossibility in Grammatical Systems. ROA-880.

Samek-Lodovici, V. and A. Prince.1999. *Optima*. ROA-363.

Samek-Lodovici, V. and A. Prince. Fundamental properties of harmonic bounding. ROA-785.

Smolensky, P. and G. Legendre 2006. *The Harmonic Mind: From Neural Computation to Optimality-Theoretic Grammar*. Cambridge,MA: MIT Press.

Tesar, B. 2007. A Comparison of Lexicographic and Linear Numeric Optimization Using Violation Difference Ratios. ROA-939.

Tesar, B. and A. Prince. 2003. Using phonotactics to learn phonological alternations. ROA-620 & *Proceedings of the Thirty-Ninth Conference of the Chicago Linguistics Society*, Vol. II.

Wikipedia. 2007. Norm_(mathematics).http://en.wikipedia.org/wiki/Norm_(mathematics)

Wikipedia. 2007. Pareto efficiency. http://en.wikipedia.org/wiki/Pareto_efficiency