A Proof of the Loehr-Warrington Amazing TEN to the Power n Conjecture

Shalosh B. EKHAD[1]*, Vince VATTER*[1]*, and Doron ZEILBERGER*[1]

Theorem: There are 10^n words in the alphabet $\{3, -2\}$ of length $5n$, sum 0, and such that every factor that sums to 0 and that starts with a 3 may not be immediately followed by a -2.

Proof: 1. Download Maple package `TEN` from `http://www.math.rutgers.edu/~zeilberg/tokhniot/TEN`. Save it as `TEN`. **2.** Go into Maple, and type: `read TEN:`. **3.** Let your computer rediscover a (linear) grammar for these words, by typing `G:=DiscoverGrammar(3,2,3):` (if you actually want to see it replace the : by ;). **4.** To prove (rigorously!) that the conjectured grammar G indeed describes (unambiguously!) our language, type: `ProveGrammar32(G,3);` and get the output `true`. **5.** To get the weight-enumerator for the language (where the weight of a word w is $x^{length(w)}$), type `GFgrammar(G,x);` and get the output $1/(1-10x^5)$. $\square$ (**endproof**) $\square$ (**endpaper**)

Appendix: TEN Remarks for Human Readers

1. The Theorem was conjectured by Nick Loehr and Greg Warrington, in a more general setting, where 3 and -2 are replaced by general relatively-prime positive and negative integers, a and $-b$, $5n$ is replaced by $(a+b)n$, and 10^n is replaced by $\binom{a+b}{a}^n$. The general case is still open. Greatly inspired by our proof, Loehr and Warrington, together with Bruce Sagan, found a computer-free proof of our theorem. They then extended their approach to prove it for $b = 2$ and all odd a. This will appear in their forthcoming paper *"A Human Proof for a Generalization of Shalosh B. Ekhad's* 10^n *Lattice Paths Theorem."* We wish to acknowledge stimulating discussions with these three humans, and of course we thank Nick and Greg for conjecturing such a beautiful result in the first place, and Bruce for telling Vince, who told Doron, who told Shalosh.

2. The Maple package `TEN` that (automatically!!) discovered the grammar, and then (automatically!!!) proved its correctness, and then (automatically!) computed the weight-enumerator, was written by VV and DZ. It was executed by SBE.

3. For a blow-by-blow description of how the grammar was *discovered*, type

`G:=DiscoverGrammarVerbose(3,2,3):` .

(The output file may be viewed/downloaded at `http://www.math.rutgers.edu/~zeilberg/tokhniot/oTENdgv`.)

4. People with no access to Maple may view the output at
`http://www.math.rutgers.edu/~zeilberg/tokhniot/oTEN32t`,
`http://www.math.rutgers.edu/~zeilberg/tokhniot/oTEN32v`,

[1] Department of Mathematics, Rutgers University (New Brunswick), Hill Center-Busch Campus, 110 Frelinghuysen Rd., Piscataway, NJ 08854-8019, USA. `[ekhad, vatter, zeilberg] at math dot rutgers dot edu` , `http://www.math.rutgers.edu/~[zeilberg/ekhad.html, vatter, zeilberg]` . First version: Sept. 14, 2005. Accompanied by Maple package `TEN` downloadable from Zeilberger's website. Supported in part by the NSF.

and `http://www.math.rutgers.edu/~zeilberg/tokhniot/oTEN32vv` for the terse, verbose, and very verbose versions.

5. The best way to understand the *heuristics* behind the **act of discovery** (of the linear grammar for the language of the theorem) and the *logic* behind the **act of verification** (proving rigorously that the empirically-conjectured grammar is indeed correct) is to carefully read the **Maple source-code**, generously made available by the authors, free of charge, even though the same **methodology** is very likely to solve other problems. There are ample comments, and readers are encouraged to experiment with `TEN` themselves. A nice exercise would be to type `DiscoverGrammar(1,1,3);` for the grammar of the language of zero-sum words in $\{-1,1\}$ that avoid factors of the form $1[-1](-1)$, where $[-1]$ denotes a word that sums up to -1. Then prove its validity by human means.

6. What if you don't know Maple well enough to follow someone else's code? Then go and learn Maple! It would be time much better spent than reading esoteric papers like this one.

7. Having said that, as a concession to Maple-illiterate people, let us briefly describe the methodology of **discovery**, to be followed by a description of the methodology of **proof**.

Procedure `Corpus` first generates all the words of our language up to a specified length. For any language L in the alphabet $\{a,-b\}$, and any pair of words in that alphabet (not necessarily in L), $[w_1,w_2]$, let $L(w_1,w_2)$ be the subset of L consisting of those words of the form w_1uw_2 for some word u. We want to construct a binary *family tree*, rooted at $L=L(\phi,\phi)$, whose vertices are pairs $[w_1,w_2]$ that stand for $L(w_1,w_2)$. It may happen that $L(w_1,w_2)$ is **empty**, i.e. L has no words of the form w_1uw_2. It may also happen that such an $L(w_1,w_2)$ is a **clone** of another $L(w_1',w_2')$, i.e.

$$\{v \mid w_1vw_2 \in L\} = \{v \mid w_1'vw_2' \in L\} \quad ,$$

where $length(w_1') + length(w_2') < length(w_1) + length(w_2)$.

Each $L(w_1,w_2)$ may be naturally partitioned in two different ways.

$$L(w_1,w_2) = L(w_1a,w_2) \cup L(w_1(-b),w_2) \quad (\cup\{w_1w_2\},\ if\ w_1w_2 \in L) \quad , \qquad (HeadWay)$$

since the letter immediately following w_1 is either a or $-b$ (assuming that the middle part is not null, in which case we have to add the additional singleton) or

$$L(w_1,w_2) = L(w_1,aw_2) \cup L(w_1,(-b)w_2) \quad (\cup\{w_1w_2\},\ if\ w_1w_2 \in L) \quad , \qquad (TailWay)$$

since the letter immediately preceding the w_2 is either a or $-b$ (assuming that the middle part is not null, in which case we have to add the additional singleton).

Anthropomorphizing a bit, we can think of every vertex as a man who has two wives, let's call them Rachel and Leah. Unlike Jacob, this man doesn't decide whom he likes better until he sees the sons that they give him. Rachel gives birth to Joseph and Benjamin, and Leah gives birth to

Reuben and Simon (let's pretend that Levi et al. never got born). Now, deciding by the children's merit, he picks one of the wives as the main one and the other one becomes his concubine, and her sons get disowned. He may not mix sons! He either picks Joseph and Benjamin as heirs, or Reuben and Simon, but he is not allowed to pick, say, Joseph and Simon.

Having picked Joseph and Benjamin as Jacob's legal heirs, they are now leaves in our expanding tree. Consider such a new vertex (that starts out as a bachelor). It may be empty, or it may be a **clone** of some older (legitimate!) relative (not necessarily a direct ancestor, e.g. Abraham is definitely okay, and Isaac, of course, but Essau would also do). In that case it becomes a **permanent leaf** (confirmed old bachelor), and is **forbidden** to have sons. But if it is neither empty nor a clone, then he, in his turn, has two son-pairs and he must decide which pair to pick as legal heirs, in other words, he has to make up his mind whether to split according to $(HeadWay)$ or according to $(TailWay)$.

The way procedure `DiscoverGrammar` in our Maple package `TEN` decides this issue is by preferring those sons with congenial cardinalities, i.e. whose greatest-common-divisor with powers of $\binom{a+b}{b}$ is as large as possible. We admit that it is only one possible heuristics for picking heirs, and its only merit is that it worked. The process terminates when all the leaves are either empty or clones.

Having discovered the grammar, to find the **weight-enumerator** of its language, (this is implemented in procedure `GFgrammar` of `TEN`), we set up a system of linear equations, whose unknowns are the weight-enumerators of the $L(w_1, w_2)$'s, let's call them $z[w_1, w_2]$, for all vertices $[w_1, w_2]$ (both internal vertices and leaves). For each internal vertex $[w_1, w_2]$, we have the equation

$$z[w_1, w_2] = z[w_1', w_2'] + z[w_1'', w_2''] \quad , \qquad (Internal)$$

where $[w_1', w_2']$, $[w_1'', w_2'']$ are the two (legitimate!) children of $[w_1, w_2]$. If $w_1 w_2 \in L$, we have to use, instead

$$z[w_1, w_2] = z[w_1', w_2'] + z[w_1'', w_2''] + x^{len(w_1)+len(w_2)} \quad . \qquad (Internal')$$

For each leaf, $[w_1, w_2]$, if it is empty, we have the obvious equation

$$z[w_1, w_2] = 0 \quad , \qquad (LeafEmpty)$$

while if it is a clone of $[w_1', w_2']$, say, then we have

$$z[w_1, w_2] = x^{len(w_1)+len(w_2)-len(w_1')-len(w_2')} z[w_1', w_2'] \quad . \qquad (LeafClone)$$

Now we (or rather Maple) solve(s) this huge system of equations and get(s) the weight-enumerators of all vertices, in particular, $z[\phi, \phi]$, our object of desire.

8. We still need to **rigorously** prove that the language generated by our putative grammar is indeed the language of interest. Since the paternity part is obviously true, we have to prove leafness, both of the empty and the clone kinds. To prove that a supposedly empty leaf is indeed so, we must demonstrate that for each such (supposedly) empty leaf $[w_1, w_2]$, there can never be a *good* word (i.e. a word in L, of *whatever* length), of the form $w_1 v w_2$. In other words we have to **logically** prove the implication:

Every word of the form $w_1 v w_2$ must be bad, i.e. contain a mishap, that is a factor of the form $a[-a](-b)$. (Here, for any integer A, $[A]$ denotes any word (in the alphabet) that adds up to A.)

Note that if $w_1 v w_2$ belongs to L then the sum of v is $-A$, where $A := sum(w_1) + sum(w_2)$, so we have to prove that every word of the form $w_1[-A]w_2$ *must contain a mishap*, i.e. a factor of the form $a[-a](-b)$.

The claim that vertex $[w_1, w_2]$ is a clone of vertex $[w_1', w_2']$ is:

"$w_1 v w_2$ is good iff $w_1' v w_2'$ is good", or equivalently,

"$w_1 v w_2$ is bad iff $w_1' v w_2'$ is bad",

which really contains two statements

"if $w_1 v w_2$ is bad then $w_1' v w_2'$ is bad", and

"if $w_1' v w_2'$ is bad then $w_1 v w_2$ is bad".

To prove such inclusions, the computer looks at all *potential mishaps*, which consist of an actual 'a' in the w_1 part and a potential '$-b$' in the v part, or a potential 'a' in the v part and an actual '$-b$' in the w_2 part. Also, conceivably (but rarely) we should consider an actual a in the w_1 part and an actual $-b$ in the w_2 part. Each such potential mishap entails some factorization of v of the form $[A]a[B]$ or $[A](-b)[B]$, or in the last case, just plain $[A]$. This is implemented in procedure `PotentialMishaps` of `TEN`.

At the bottom line, proving that $L(w_1, w_2)$ is *empty* boils down to proving that every word of the form $w_1[A]w_2$, where $A := -(sum(w_1) + sum(w_2))$, **must** contain some mishap. Also proving cloneship reduces to proving several statements of the form: "$u_1[A]u_2[B]u_3$ **must** always contain some mishap", where u_1, u_2, u_3 are *specific words*, and $[A], [B]$ denote arbitrary words that sum to A and B respectively, for some specific integers A and B. Of course, w.l.o.g. the $[A]$ and $[B]$ words are mishap-free.

A crucial tool in the automated proof of such assertions is a recurrence that is a simple consequence of the **Discrete Rolle Theorem**:

$$[A] = \begin{cases} \displaystyle\bigcup_{i=1}^{a} [i-a]a[A-i], & \text{if } A > 0; \\ \displaystyle\bigcup_{i=1}^{b} [-i+b](-b)[A+i], & \text{if } A < 0. \end{cases} \qquad (WordRecurrence)$$

Indeed any word that sums to $A > 0$ must have its **shortest** prefix whose sum is positive, the last letter of that prefix being, necessarily, a (or else it wouldn't be the shortest), and the sum of that prefix (by 'continuity'!) must be between 1 and a (inclusive). Analogously, for $A < 0$.

By iterating these recurrences it is easily seen that each word $[A]$ can be written as one of several *factorizations* featuring the letters a and $-b$ and the **fundamental factors** $[0], [1], \ldots, [b-1]$. We can even apply $(WordRecurrence)$ to these, but then we'll get 'self-referential' expressions, i.e. ways of expressing them in terms of themselves (and the actual letters a and $-b$) as a kind of 'Chomskian' grammar.

Whenever we have to prove emptyness or cloneship, we have to prove that a certain set of words doesn't exist. Like in 4CT or any *reducto* proof (all the way back to $\sqrt{2} \notin Q$), if the set was not empty, there would be a **minimal counterexample**. So it suffices to prove that minimal counterexamples do not exist. Note that a minimal counterexample (in our business) can never have a *proper* zero-factor (removing a zero-factor from a good word obviously results in another good word). So if we are in the lookout for minimal counterexamples, we can considerably reduce the many options that an $[A]$ word can have, by discarding those factorizations in the enlarged ('meta-') alphabet $\{a, -b, [1], [2], \ldots, [b-1]\}$ that contain a zero-sum proper factor. Of course we also kick out any such meta-word that contains a mishap, i.e. a factor of the form $a[-a](-b)$. If there are any survivors left after the first purge, we can replace $[1], \ldots, [b-1]$ by their self-referential expressions getting longer 'meta-words', and once again discard all those that contain mishaps and all those that contain a zero-sum proper factor. If all goes well, nothing will be left after finitely many purges. For the $\{3, -2\}$ case three purges sufficed to prove every instance of emptyness or cloneship.

Let's explicate the above for our case of interest, $a = 3$, $b = 2$. There is only one 'meta-letter', $[1]$, and we have, according to `TEN` (but this is so simple that even you can do it!)

$$[1] = \{[3, -2], [-2, 3], [-2, [1], 3, [1], -2]\} \quad ,$$

which means that every word in the alphabet $\{3, -2\}$ with no zero-sum-factors and no mishaps and whose sum is 1, is either the two-letter word $(3)(-2)$ or the two-letter word $(-2)3$ or else can be written as $(-2)w_1 3 w_2(-2)$, where w_1, w_2 are zero-sum-free and mishap-free words whose sum is 1 on their own right.

We also have:

$$[-1] = \{[-2, [1]], [[1], -2]\} \quad ,$$

stating that any mishap-free and zero-factor-free word in $\{3, -2\}$ that sums up to -1 may be written as $(-2)w_1$ or $w_1(-2)$ where w_1 is a such a good word that adds up to 1. You are welcome to decipher the following lemmas that Shalosh discovered and proved.

$$[-2] = \{[-2], [[1], -2, -2, [1]]\} \quad ,$$

$$[2] = \{[3, -2, [1]], [3, [1], -2], [-2, 3, [1]], [-2, [1], 3], [[1], -2, 3]\} \quad ,$$

$$[3] = \{[3], [-2, [1], 3, [1]], [[1], -2, 3, [1]], [-2, 3, 3, -2, [1]], [-2, 3, 3, [1], -2], [-2, 3, -2, 3, [1]]\} \quad .$$

9. The grammar is especially simple for the case $b = 1$. If $A_a(n)$ is the set of good words in $\{a, -1\}$ we have (here $\overline{1}$ stands for -1)

$$A_a(n) = A_a(n-1)\, a\, \overline{1}^{a} \;\cup\; \overline{1}\, A_a(n-1)\, a\, \overline{1}^{a-1} \;\cup\; \overline{1}\,\overline{1} A_a(n-1) a\, \overline{1}^{a-2} \;\cup \ldots \cup\; \overline{1}^{a}\, A_a(n-1)\, a \quad ,$$

immediately implying that $|A_a(n)| = (a+1)^n$.

10. "Mr. Watson, come here. I want you!"

While grammatically correct, this sentence is not quite Shakespeare. Its great significance is the **way** it was uttered, via the telephone, by Alexander Graham Bell. Analogously, while our proof is definitely closer to Gauss than Bell's sentence is to Shakespeare, its main significance, if we do say so ourselves, is in the **way** that it was discovered, and especially the **way** that it was **proved**, all by computer! The role of the humans (VV and DZ) is no longer that of *athlete*, but that of *coach*, no longer *prover* but *programmer*, or if you wish, *meta-prover*. We believe that this will be **the way to go** sooner than you think! Amen.