# Tree Notation: an antifragile document notation

Breck Yunits and a,a,b,b,c,c,d,d,j,j,m,m,m,n,n,p,r,s,t,t,x

***Abstract*—This paper is written for programmers around the world. We hope by freely sharing a powerful new discovery, we might inspire you, dear programmer, to create new tools and languages, so that we all may experience a quantum leap in programming productivity.**

**We include a Visual Abstract to succinctly display the problem and discovery. Then we describe the problem–the abstract syntax tree (AST)–and introduce the novel solution we discovered: a new family of 2D programming languages that align source code with geometric trees and remove the AST. Then we make some predictions and conclude with a plea to steal this idea.**

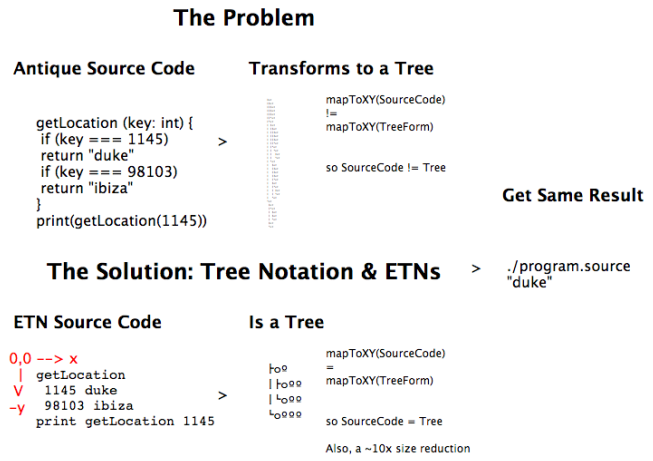


Fig. 1. This Visual Abstract explains the core idea of the paper. This diagram is also the output of a program written in a new ETN called Flow.

## I. The Problem

Programming is complicated. Our current high-level antique languages (HALs) add to this complexity. To run HAL code we first transform it to an AST and then transform that to executable machine code. ASTs have enabled local minimum gains in developer productivity. But programmers lose time and insight due to discrepancies between HAL code and ASTs. We searched, over a number of years, to discover a simpler way, and stumbled upon Tree Notation (TN).

## II. The Solution: Tree Notation

In this paper and accompanying GitHub ES6 Repo (GER - github.com/breck7/treenotation), we introduce TN, a new 2D notation. A node in TN maps to an XY point on a 2D plane. You can extend TN to create domain specific languages (DSLs) that don't require a transformation to a discordant AST. These DSLs, called ETNs ("Extends Tree Notation"), are easy to create and can be simple or Turing Complete.

Breck Yunits is a software engineer from Brockton, MA (breck7@gmail.com).

TN encodes one data structure, a **TreeNode**, with two members: a string called **line** and an optional array of child TreeNodes called **children**.

TN defines two special characters, Y Increment (YI) and X Increment (XI). YI is "\n" and XI is " ". XI is a space, not a tab. Many ETNs also add a Word Increment (WI) to provide a more succint way of encoding common node types.

A comparison quickly illustrates nearly the entirety of the notation:

JSON:

```
{
 "title" : "Jack and Ada at BCHS",
 "visitors": {
  "mozilla": 802
 }
}
```

Tree Notation:

```
title Jack and Ada at BCHS
visitors
 mozilla 802
```

## III. Useful Properties

### A. Simplicity

As shown in Fig 1, TN simply maps source code to an XY plane which makes reading code locally and globally easy, and ETN programs use far fewer source nodes than equivalent HAL programs. TN lets programmers do more with less.

### B. Zero parse errors

Parse errors do not exist in TN. Every document is a valid TN document. ETNs implement microparsers that parallelize easily and can creatively handle errors.

With most HALs, to get from a blank document to a certain valid document in keystroke increments requires stops at invalid documents. With TN all intermediate steps are valid.

A user can edit the nodes of a document at runtime with no risk of breaking the TN parsing of the entire document. "Errors", still can exist at the ETN level, but ETN microparsers can handle errors independently and even correct errors automatically. ETNs slapchop development and debug time.

### C. Semantic diffs

HALs ignore whitespace and so permit programmers and programs to encode the same object to different documents. This often causes merge conflicts for non-semantic changes.

TN does not ignore whitespace and diffs contain only semantic meaning. Just one right way to encode a tree.

### D. Easy composition

Base notations such as XML,[2] JSON,[3] and Racket[4] can encode multi-lingual documents by complecting language blocks. For example, the JSON program below requires extra nodes to encode Python:

```
{
 "source": [
  "import hn.np as lz\n",
  "print(\"pdm mo gb 28-3\")"
  ]
}
```

With TN, the Python code requires no complecting:

```
source
 import hn.np as lz
 print("pdm mo gb 28-3")
```

## IV. Drawbacks

TN is new and tooling support rounds to zero.

TN lacks primitive types. Many HALs have notations for common types like floats, and parse directly to efficient in-memory structures. ETNs make TN useful. The GER demonstrates how useful ETNs can be built with just a few lines of code, which is far less code than one needs to implement even a simple HAL such as JSON.[5]

TN is verbose without an ETN. A complex node in TN takes multiple lines. Base HALs allow multiple nodes per line.

Some developers dislike space-indented notations, some wrongly prefer tabs, and some just have no taste.

## V. Predictions

**Prediction 1: no structure will be found that cannot serialize to TN.** Some LISP programmers believe all structures are recursive lists (or perhaps "recursive cons"). We believe in The Tree Conjecture (TTC): **All structures are trees**.

For example, a map could serialize to MapETN:

```
ty aaaabbcccdddefgggghjjjjjkllmmmnopprrsssvz
0106 9de7a5727e35ebfccc5937459f62042bc15d8a35
```

Therefore, maps are a type of tree. TTC stands.

**Prediction 2: ETNs will be found for every popular AL.** Below is some code in a simple ETN, JsonETN:

```
o
 s dsl yrt
 n ma 902
```

ETNs will be found for great ALs including C, RISC-V, ES6, and Arc. Some ETNs have already been found,[6] but their common TN DNA has gone unseen. The immediate benefit of discovering an ETN for an AL is that programs can then be written in an ETN editor and compiled to that AL.

**Prediction 3: Tree Oriented Programming (TOP) will supersede Object Oriented Programming.** A new style of programming, TOP, will arise. TOP programmers will frequently reference 2D views of their program.

**Prediction 4: The simplest 2D text encodings for neural networks will be ETNs.** High level ETNs will be found to translate machine written programs into understandable trees.

## VI. Literature Review

Turing Machines with 2D Tapes have been studied and are known to be Turing Complete.[7] A few esoteric 2D programming languages are available online.[8] At time of publication, 312 programming languages and notations were searched. Over 100,000 pages of research was read. TN was not found.

The closest someone came to discovering TN, and ETNs, was perhaps Mark B. Wells, who wrote a brillant paper at Los Alamos, back in 1972 which predicted the discovery and utility of TN and ETNs. Alas, his paper went almost entirely ignored.[1] This exemplifies why they call it REsearch.

## VII. Steal this idea

We now turn our attention to building new ETNs and tools for data science, visual programming, ML, parallel computing and more. The GER contains a TN library, some ETNs, and one more thing. We hope you, fellow programmers, will "steal" this idea and together we can use this new discovery to build faster!

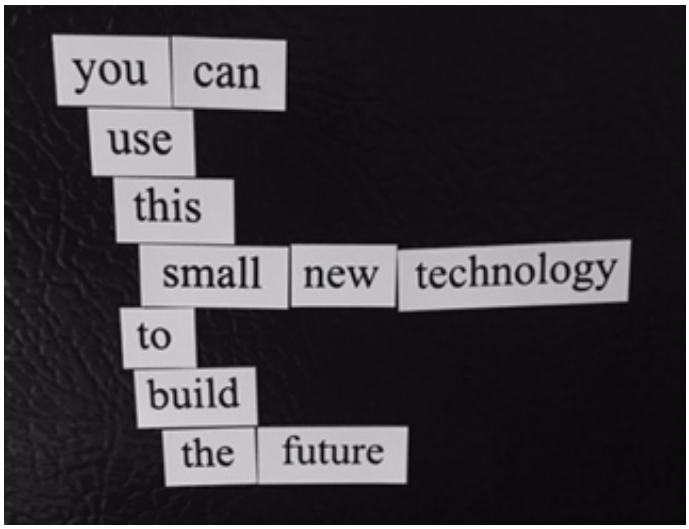


Fig. 2. Rearranging these fridge magnets is equivalent to editing a TN document. The fridge magnet set that includes parentheses is a poor seller.

## References

[1] Wells, Mark B. "A REVIEW OF TWO-DIMENSIONAL PROGRAMMING LANGUAGES*". 1972. http://sci-hub.cc/10.1145/942576.807009 (visited in 2017).

[2] Bray, Tim, et al. "Extensible markup language (XML)." World Wide Web Consortium Recommendation REC-xml-19980210. http://www. w3. org/TR/1998/REC-xml-19980210 16 (1998): 16.

[3] Crockford, Douglas. "The application/json media type for javascript object notation (json)." (2006).

[4] Tobin-Hochstadt, Sam, et al. "Languages as libraries." ACM SIGPLAN Notices. Vol. 46. No. 6. ACM, 2011.

[5] Ooms, Jeroen. "The jsonlite package: A practical and consistent mapping between json data and r objects." arXiv preprint arXiv:1403.2805 (2014).

[6] Roughan, Matthew, and Jonathan Tuke. "Unravelling graph-exchange file formats." arXiv preprint arXiv:1503.02781 (2015).

[7] Toida, Shunichi. "Types of Turing Machines." http://www.cs.odu.edu/ toida/nerzic/390teched/tm/othertms.html (visited in 2017).

[8] Ender, Martin. "A two-dimensional, hexagonal programming language." https://github.com/m-ender/hexagony (visited in 2017).