

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.

Notation”), are easy to create and can be simple or Turing Complete.

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 succinct 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. See more and do much more with much 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 slash development and debug time.

C. Semantic diffs

HALs ignore whitespace and so permit programmers and programs to encode the same object to different documents. Sometimes this helps but more often this causes merge conflicts for small or non-existent semantic changes.

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

The Problem



The Solution: Tree Notation & ETNs

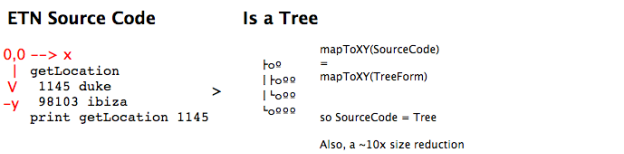


Fig. 1. This Visual Abstract explains the core idea of the paper. This diagram is also the output of a program written in a new ETN called Flow.

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 whitespace-based 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

D. Easy composition

Base notations such as XML,¹ JSON,² and Racket³ can encode multi-lingual documents. But to do that, those notations often complicate those blocks.

In the example below, the program IPython encodes Python to JSON. The resulting document contains additional nodes:

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

With TN, the Python block is indented and requires no completing:

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

IV. DRAWBACKS

A. Lack of Tooling and Support

TN is new, and library and application support, compared to other popular base notations, rounds to zero.

B. Lack of Primitive Types in TN

Some HALs have notations for common primitive types like floats, and parse documents directly to efficient in-memory structures. "TN" is minimal so for best results, don't forget about "E" (to get "ETN").

The need for ETNs is a minor concern, as the GER demonstrates how useful ETNs can be built with just a few lines of code. And creating a TN library in other HALs is straightforward, and not comparable to the complexity of implementing a JSON library,⁴ for example.

C. Aesthetic Differences

Without an ETN, TN can be verbose. A complex node in TN extends over multiple lines. HALs have a denser display of information out of the gate, with multiple nodes per line.

Other concerns have been raised. 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 data structures are recursive lists (or perhaps "recursive cons"). After seeing TN, we believe in The Tree Conjecture (TTC): **All structures are trees.**

For example, a map could serialize to MapETN:

```
ty aaaabbbccdddefggghjjjjklmmnnopprsssvz
106 3cf5d9a07b7a51e1324e4accd1cd6faf71ce7c48
```

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

Prediction 2: Useful new ETNs may be found for every AL. Below is some example code in a simple ETN, JsonETN.

```
o
s foundOn Sci-Hub
n ma 902
```

ETNs will be found for great ALs including C, RISC-V, ES6, and Arc. Some ETNs have already been found,⁵ but haven't been recognized as ETNs. 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 a tree view of their program.

Prediction 4: The simplest 2D text encodings for neural networks will be ETNs. High level ETNs will be found to transform massive GPU trained routines into understandable trees.

VI. 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!

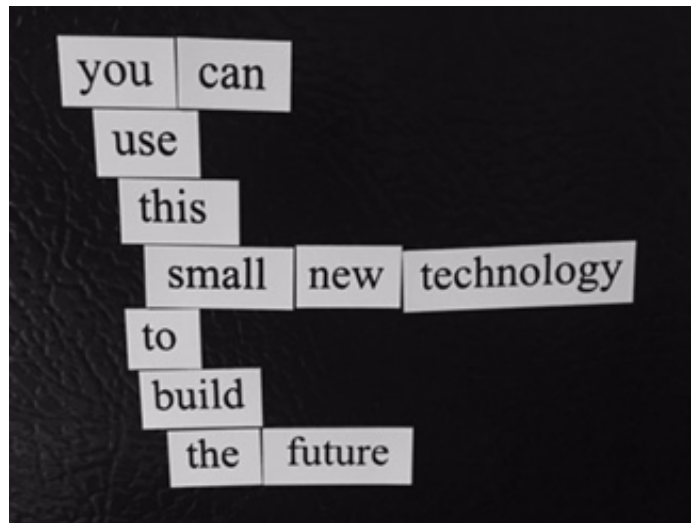


Fig. 2. Rearranging these fridge magnets is equivalent to editing a TN document. The fridge magnet set that includes parentheses is a poor seller.

REFERENCES

- 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.
- Crockford, Douglas. "The application/json media type for javascript object notation (json)." (2006).
- Tobin-Hochstadt, Sam, et al. "Languages as libraries." ACM SIGPLAN Notices. Vol. 46. No. 6. ACM, 2011.
- Ooms, Jeroen. "The jsonlite package: A practical and consistent mapping between json data and r objects." arXiv preprint arXiv:1403.2805 (2014).
- Roughan, Matthew, and Jonathan Tuke. "Unravelling graph-exchange file formats." arXiv preprint arXiv:1503.02781 (2015).