Virtual Muse

Charles O. Hartman

Published by Wesleyan University Press

> > > > / 7 /

PROSE

The next programming project I'll present is in some ways the climax of this book. It's the longest in sheer number of program lines. And while though the idea behind it isn't original, the program does more to *originate* poetry than any of the others.

Yet it represented a kind of retrenchment after the elaborations of AutoPoet. Conceptually, what I did was to remove the iambic pentameter filtering of output text. The remaining program functions coordinated the contents of a dictionary and a grammar, each of them a text file that could be edited separately. The program now produced a sequence of grammatically correct English sentences. Feeling chastened, I called it Prose.

By the time I got to Prose, the dictionary-and-grammar-handling routines had been through at least four major revisions in Natural Section and AutoPoet. I had rewritten them in three different computer languages (Prolog, Pascal, and C). By now they worked smoothly.

Prose offers a "show tree" option. I built this in to help me in debugging the program while I wrote it. But it also exhibits some interesting information in its own right. If this option is turned on, the output looks something like what is shown in figure 2.

Everything between the "=tree=" and "=end=tree=" markers shows the program at work building the syntactical template. Then it builds the actual sentence (about "the order of harm") by randomly selecting words of the right types from the dictionary. To see in more detail how Prose works, let's look at some pieces of this "grammar tree."

The metaphor of a tree is used in linguistics as well as in program-

```
=tree=======================
SENTENCE:
        DEPCLAUSE:
                —>SubordConj
                NOUNPHRASE:
                        —>!the
                        —>Noun
                        —>!of
                        —>#PushPlur
                        —>Substance
                        —>#PopPlur
                VERBPHRASE:
                        INTRVBUNIT:
                                —>IntrVerb
        —>@,
        INDCLAUSE:
                NOUNPHRASE:
                        —>PossPron
                        —>Noun
                VERBPHRASE:
                        TRANVBUNIT:
                                —>TransVerb
                        OBJPHRASE:
                                NOUNPHRASE:
                                        —>Substance
        —>@. [14 tokens in sentence template]
====================end=tree=
```

If the order of harm fought, my question experienced art.

Figure 2

ming. (This is no accident, but what the coincidence means depends on whether you ask a programmer or a linguist.) If you turned this diagram of a sentence on its side and skewed it picturesquely, it could be seen as having a trunk and a lot of branches terminating in leaves (figure 3).

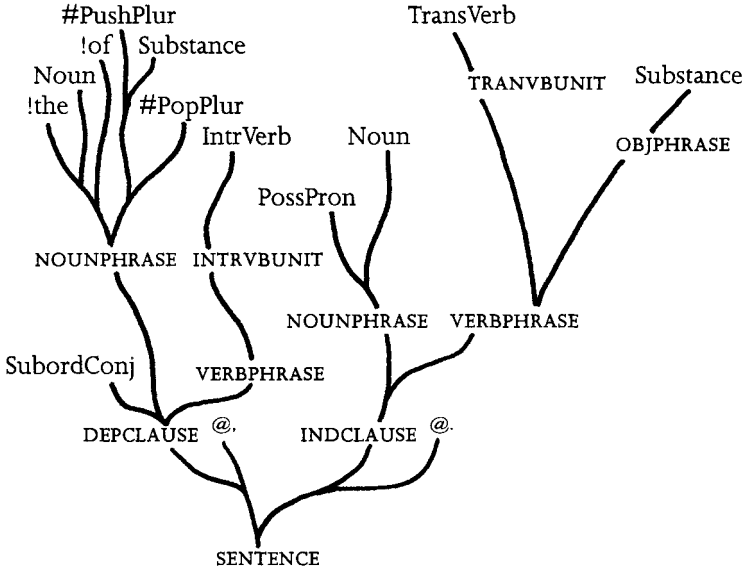

Figure 3

The trunk is the basic unit, Sentence. The process always begins with that. The leaves are the components that finally go into the sentence template itself. In the program's version of the tree diagram, printed earlier, each leaf is indicated by an arrow: —>. So Sentence doesn't become part of the template, but "SubordConj" (for "subordinating conjunction") does, along with "!the" and "@,".

Some of these template units begin with signs that indicate special functions. The "@" indicates punctuation—here, the comma in the middle of the sentence and the period at its end. The "!" indicates a *literal*, that is, a word that's to be inserted into the sentence as it is, rather than a *type* of word to be looked up in the dictionary. Literals are especially useful for placing whole phrases in the grammar, like "the A of B," which might be encoded, "!the Noun !of Noun." This is often easier than encoding the proper treatment of phrases (especially idioms) in the program itself.

A "#" indicates a *flag*—a signal to the sentence-building routines to perform some special task. These flags get complicated in special-

ized ways, but I'll give a quick explanation of the two that appear in the sample template, "#PushPlur" and "#PopPlur."

Think for a moment how plurality (number) works in sentences. Number gets established somehow—for instance by the subject of the sentence, a noun that is either singular or plural. When we get to the verb, we have to remember the number and make the verb agree with the subject. If we began with "The dog," we have to continue with "dances," not "dance." In between the subject and the verb, though, might come another phrase ("whose teeth are shining"). This phrase too has to be internally consistent in number (not "teeth is"). But no law says it has to be the same in number as the clause it's interrupting. So as we're speaking, inventing the sentence, we *suspend* the state of plurality for a moment, then *restore* it when the digression is over.

In other words, plurality works like the data structure known to programmers as a "stack." "Pushing" and "popping" are the operations that put something onto a stack and take something off it. The usual metaphor is of cafeteria plates in a spring-loaded dispenser. The program suspends plurality by "pushing" the current state (singular or plural) onto the stack. Then a new plurality can be established and govern its phrase without destroying the old information. When the phrase is over, we "pop" the old plurality off the stack to make it active again, ready for the upcoming verb. Sometimes there are several nested levels of plurality in a sentence. ("The dog, whose teeth shine as the moon does, dances.")

The program itself knows only words, not phrase structures. So the grammar (which is kept in a separate text file for easy editing) has to control shifts in the level of plurality. "#PushPlur" and "#PopPlur" are the flags that let it do that. Other flags do similar jobs of passing messages from the grammar to the program.

A program really smart about human language would know about things like phrases. It would have a *memory* for suspended items like plurality. Putting this information by hand into the grammar is cheating, as far as AI programming is concerned. But I wasn't concerned about purity in the programming. I just wanted to produce unpre-

dictable but fruitful linguistic material. The fact that I would renounce laziness in the poetry, and not in the programming, just confirmed my sense of proportion and values.

Let's look back at the sentence tree I showed earlier, the odd horizontal one the program prints out. The horizontal indentations in it stand for the layers of a syntactical hierarchy. Each vertically aligned group corresponds to one of the rules in the grammar file. For instance, the Sentence shown earlier is made up of four parts: a dependent clause, a comma, and an independent clause followed by a period. In the grammar, this rule is written:

SENTENCE/ DEPCLAUSE @, INDCLAUSE @.

When the program prints a tree, it arranges these four elements one above the other.

The DepClause, in turn, is made up of a SubordConj, a NounPhrase, and a VerbPhrase. The hierarchy continues until all the constituents of every level are "leaves" that aren't defined by further "branches." The formal way to express this exhaustive nesting of rules depending on other rules is to call the process *recursive*. Recursion is important in computing; it's also essential to the structure of human languages. We all speak sentences made up of clauses made up of phrases made up of other phrases made up of words. (Only the words come out of our mouths, but the phrases and clauses really exist.) Linguists have shown that we do this by applying grammar rules recursively.

Another sentence might be just an IndClause and a period: "SENTENCE/ INDCLAUSE @." Still another might be an IndClause followed by a semicolon and another sentence. Other sentences are questions, which require different forms from declarations. The grammar contains many rules that define sentences.

Similarly, there are two kinds of VerbPhrase in this sentence: one intransitive (made up of an intransitive verb unit, IntrVbunit—in this case, simply an intransitive verb) and one transitive (made up in this case of a transitive verb unit and an object phrase). In the tree these verb phrases are shown as further indented vertical groups.

When one rule (like the one that defines Sentence) calls for an-

other rule (like one that will define a VerbPhrase), the program chooses *at random* among all the rules of that type in its grammar—all the rules that define the part being called for. This sentence needed two VerbPhrases, and randomly chose different rules for them. If the grammar contains six VerbPhrase rules, each will be chosen about one-sixth of the time. Originally, the odds could be loaded by repeating a favorite rule several times in the grammar. Later I added a "weight" factor to each rule for greater efficiency.

A side note: I've said before that human choices can be arbitrary but not random. And even the "random" numbers in computer programs aren't truly random, like the decay times of elementary particles. They are generated by functions that produce an unpredictable sequence but that, beginning from the same "seed" number, will reproduce the same sequence. The sequence repeats after a certain point—usually, the largest number available in some unit of computer storage, often 65,536. My recent Macintosh-based MacProse uses this fact to advantage. The program usually displays the tree of the most recently generated sentence. When the user clicks the mouse on an earlier sentence, the program reconstructs the tree to display by reseeding the random-number generator with the original value and running its sequence up to the point where the sentence was produced. That way, it has to store only the seed value and access count for each sentence, rather than the very bulky tree itself.

There's no limit to how many levels there can be in the syntactical hierarchy of a sentence. (Technically, there's a limit imposed by the size of the program's stack in memory, which can get overfilled by a recursive function.) There is a practical limit on how big the template can be (though it could be expanded if necessary). With the grammars I've used, about sixty items are the most that ever show up in the template. A sixty-word sentence is unusual, though far from a record. (Writing a clear sentence of one hundred words is a good exercise.)

Randomness comes in again when the program takes its finished template off to the dictionary and asks for a particular noun or conjunction or possessive pronoun. From one single template, therefore, the program might produce thousands of different sentences.

And from one brief grammar (the one I used had well under a hundred rules) it can produce thousands of different templates. There's no worry about Prose repeating itself.

Often the words can't be used just as they come from the dictionary. If the sentence calls for a plural noun, for instance, it's easier to make the randomly chosen noun plural than to keep going back to the list over and over until a plural noun happens to turn up. Easier—but not always easy enough. *Car* is simple to pluralize. *City* and *tax* are harder. *Child* is impossible to pluralize by rule, so *children* is in the dictionary.

It would be especially inefficient to store all the different forms of each verb. The tree I've shown here calls for a transitive verb. But "TransVerb" isn't one of the categories in Prose's dictionary. Instead, the program fetches a transitive *infinitive* at random, such as *hoodwink*. Then, taking into consideration the tense, person, and number that may have been established by earlier events in the sentence, it makes the appropriate form of that infinitive: *hookwinked* or *hoodwinks* or *hoodwinking*.

By now it should be no surprise that Prose will produce on demand a large number of very foolish sentences. But to give it its due, it does a couple of things right. First, though some of its products seem quite strange, they are all grammatically correct, with all the urging toward sense that that implies. Second, as I've indicated, it can crank out an enormous *variety* of sentences, of any degree of complexity you care to tell it about in the grammar. The program's main virtue is its flexibility. If you don't like the kinds of sentences it makes, you can change the rules it uses just by editing its grammar file, without having to change the program itself.

Of course there are limits to this flexibility. The program embodies many assumptions about English parts of speech. It would never work with Hebrew or Hopi. As another example, I've distinguished between Nouns and Substances. There are several operational differences, but the simplest is that a Substance is a noun that's often not preceded by an article. *Tree* is a noun, *bark* is a substance. We say "an artichoke" but not "a steam," "speed kills" but "the speeder kills."

This syntactical distinction gets tangled up in semantic ones. We say "I like beef" but not "I like cow." "I like flounder" runs both ways.

The grammar I used is merely one working version. It stresses questions because I've found that they have an especially evocative effect on the reader. (Ron Silliman explored this effect in "Sunset Debris.") There are no rules that use relative pronouns, though there are relative pronouns in the dictionary. Any page of any book will provide many examples of kinds of sentences that prose can't produce because I haven't given it the rules. But it's flexible.

The same applies to the dictionary. The five thousand or so words in the present version (perhaps a quarter of them duplicates or near-duplicates for various purposes) are the result of a long evolution.

The original dictionary of five thousand common words that I gleaned from the word-frequency list for Natural Selection and AutoPoet turned out to have important flaws. Careful autopsy shows that AutoPoet's terminal boredom was partly due directly to vocabulary. After all, we need to remember where Kuçera and Searle got their corpus of a million words. Much of the English prose they sampled was recent journalism. Some of the material was literary—that is, insistently linguistically interesting. But that material constituted the same small proportion as literature does in the daily use of language. Its effects were swamped just as thoroughly as literature is in the world of printed words. Certainly among the most frequent five thousand words (out of over thirty thousand different words in the million-word corpus), not much was likely to crop up that would testify to the poetry inherent in the American soul. Most of what AutoPoet said sounded as though it came out of a committee.

So I cut words that felt like irredeemable bureaucratese: *accordance, recommendation, facilities, nonspecific, marketing.* In general, any words that pushed a sentence too hard toward abstractness were better omitted: *personality, negative, growth, velocity, location, intervention,* and dozens of others ending with *-tion.* But also, inappropriately concrete words had to go, such as most names. "Dave" and "Orleans" are among that first five thousand, but they don't help the reader's sense of *focus* in random prose. They're just disorienting.

Some good words, especially verbs, have especially tricky syntactical implications and were best quietly omitted. *Urge* requires a complicated object ("urge A to do B"). Many verbs (*admit, prove, insist*) take a "that" construction, which isn't simple to put into the grammar. *Alike* is usually the complement of a plural copula ("the brothers are alike"); to use it I would have to build in special flags that would keep that whole construction "in mind" until completed. So I dumped those.

Cutting this way brought the list down to a thousand. Then I added words I hoped would have positive effects on a reader's sense of coherence or purpose in the sentences. I began with concrete nouns: *elephant, Bebop, calico, muffin, pewter, clarinet, oak*. Earlier I had gathered for other purposes a special lexicon of words derived from poems I was working on. Many of these words—*checkmate, Babbage, metabolism, Turing, computation*—would serve.

If, as Coleridge said, "poetry is the best words in the best order," then the poet must be a specialist in recognizing "the best words." The pleasure I took in this part of the work was related to the pleasures of poetry. "Best," as sly old Coleridge knew, is a tricky and contingent measure.

Finally, I ran down a list of words I'd gotten from a researcher at Kurzweil Applied Intelligence (KAI) while I worked there as a technical writer. KAI is a major company working on the problem of speech recognition. The words are ones that present special challenges to an automatic recognizer, and the whole group of about two hundred covers the field of English phonemes very thoroughly. Like any set of words chosen for their sound, they form a workable poetic diction. Many sentences that Prose produced benefited from *asterisk, gung-ho, weed, typhoid, sleuth*, and a few dozen others.

There was a question of how many words to add. The obvious way to imitate human speech is to duplicate human vocabulary, but that was just the ground on which AutoPoet had notably failed. The opposite extreme had been used by very early computer poetry experimenters. As *Time* reported (May 25, 1962) about the Auto-Beatnik project, "By drastically cutting down [the program's] choice

of words—so that the incidence of a subject word reappearing is greatly increased—engineers can make the machine seem to keep to one topic." (Hugh Kenner used this item from *Time* as the epigraph to his essay, "Art in a Closed Field," published in *Virginia Quarterly Review* late the same year.) But this kind of single-mindedness soon comes to seem merely obsessive—human, but irritating. It was a little like the old Basic English project, six hundred words that would give the third world a common English pidgin. That wasn't what I wanted. I found that a reasonably balanced dictionary of one to two thousand words worked all right, and one of about five thousand gave a pleasant breadth.

Now I had my program; but what to do with it? The first thing I tried was the easiest. I let it run for a while and then combed through the output looking for interesting chunks I could string together. But this approach held onto a residue of my earlier false assumption. I was still treating the computer as a retarded or psychotic human brain from which I could hope for flashes (however far apart) of ordinary or extraordinary lucidity.

At first I saw no alternative. For days, sitting on the train commuting to and from classes, I kept poring over fanfolded piles of computer paper, searching in vain for oracular truths. Jorge Luis Borges has described just this situation in his story "The Library of Babel." He imagines a library of innumerable books of a certain length filled with all the random combinations of letters, in identical rooms stretching toward infinity in all directions. The narrator of the story remembers someone once finding, amid all the megatons of garbage, the phrase "O time thy pyramids." This discovery was a shining moment in the library's dreary, endless history. I was far better off. My dictionary and grammar excluded all but a tiny fraction of the possible combinations. But a tiny fraction of infinity is still infinity; and if "infinity" isn't technically involved (there are only so many possibilities), it might as well be.

Yet there were endless tempting sentences, perhaps one in five or ten: "The court of color (radiation of the center) is stress above any building." Nonsense, yes, but with the subliminal promise of an

image: open air, surrounding white buildings, uncanny color. Take out "stress," which is abstract in this context. Notice that "color" makes "radiation" unnecessary (though the connection between them may have first called my attention to "color"). And "court" (as in "courtyard") might contain the implications of both "center" and "building" and made those words unnecessary. So "The court of color is . . ." what? Air, really, or all the air considered as a whole: "atmosphere." "Atmosphere" might also be the courtroom of colors, judiciously discriminating near from far (as in aerial perspective), bright from dim.

But "atmosphere" could never have been produced by the program, not being in its dictionary. *So*, came the subversive voice, *add it to the dictionary*. I was as determined not to cheat as when I selected Travesty outputs for "Monologues of Soul and Body." But I was getting sneakier, too.

So began my work on the second poem reprinted in the Appendix. What I discovered was that I could reverse the process of "Monologues." Instead of feeding my poetry to the computer to digest (or indigest), I could alter its impromptu output to suit my own poetic sense. And what I was doing while I edited this text—the way I could hear myself thinking—felt very much like the way I think when writing poems. Most of any writing process is actually rewriting. Many writers find that the first draft is almost useless in itself. It contains just enough of truth to make the final work, however difficult, possible.

Prose, then, could be treated as a first-draft writer. Many sentences had to be ejected outright. "How was language under volume of the hotel leaving?" presented no foothold to my imagination. "I was evening of the school" didn't set any bells resonating when I came across it. A few sentences slipped through unaltered: "Where is this theory walking?" Others needed only the slightest touch. "Any spirit near man: a town" became "Any spirit near man likes a town," which among other things seems true. (Once again, I was changing the sentence only to something the program *could* have produced.) I found myself on unexpectedly firm ground. All I had to do as editor

was to give the outpost a good shake until it settled into place as sense—and keep my ears open for that sense.

So this random output—

> The court of color (radiation of the center) is stress above any building. Light inside the spring marched, but I am place of the science. Since metabolism is typing me, the oak throughout brick has worked. A party up steel: the shot of time. What is a church increasing? Before I have made us, the voice of woman (a dark dark) was numbering earth, and an easy sea does.

—rather quickly became this language I felt I could stand behind:

> The court of color is atmosphere. Light in the spring marches, but place is the true science. While metabolism types us, the oak has worked through brick, and the breath knows ghosts. Before creation, the voice of woman (a dark dark) was numbering earth, as an easy sea does.

I liked the historical sweep, the balance between nostalgia and admonition, the modulation from one metaphor to the next. Versions of light ("color" and "spring," and "a dark dark") mingle with opaque solidities like "oak," "brick," "earth," maybe "place." The slow "oak" inexorably forcing its way "through brick" stands against the faster "breath" and "voice" and "march" of the light. Yet everything is changing and alive, merely exhibiting different "types" of "metabolism." Linking all the images is the "sea," both restless and "easy," clear and dark, inanimate yet sentient enough to "number" the "earth" it washes, grain by grain.

One interesting point in the final version is the addition of "and the breath knows ghosts." I wrote that, not the program. Yet it's not a phrase I could imagine myself finding, except under the spell of the program's language, so dreamily detached from the immediate necessities of saying things. But each breath we take includes atoms breathed by Bach and Caligula. If the breath doesn't know ghosts, what does it know?

Several questions lingered, nagging. *Was* this fiddling with the computer's output really cheating? And if coherent meaning suffused

"paragraphs" like this, who was making it? Is the paraphrase I spelled out a moment ago due to the poet or to an overingenious critic?

Yet these questions finally answered each other. I wasn't doing artificial intelligence research but writing poems. And I wasn't trying to imitate a human poet. The point of my work wasn't the power or originality of the program itself. (Later I discovered—with no surprise—that about the same time I was writing Prose, Chris Westbury, in Montreal, was producing the freeware program McPoet, which seems to use the same familiar principles of a "context-free grammar.") The point, rather, was seeing how to use what it could do.

I was taking seriously the lesson I found in some of the most intriguing poetry of the present time: to let the play of language stand on the stage of the page in its own ordinary mysteriousness, encouraging the reader not just to participate in making sense but to be conscious of participating. I'd been preparing for some time (as a critic, as a reader, as a poet) to write in a new way. Several decades of thinking about poetry taught me what to ask from the computer. And the stimulus of those random sentences, in turn, brought home the possibilities I'd glimpsed. My programming and my poetry writing were at last teaching each other.

The criteria for *sense* that my ear set in the course of (re)writing would become the reader's necessary criteria, too. And that, I had come to realize, is how the writing of any poetry works. The poet sets the rules that stake out a territory in the endless realm of language, trying not to go so far out as to lose the reader's natural trust. As readers, we revel in our meaning-making ingenuity. This makes us want to trust the poem. As Wallace Stevens put it, "Poetry must resist the intelligence almost successfully." I had always known this, but the experiments were bringing the point home with a new directness.

I kept collecting and editing sentences until I knew (though it would be hard to say how) that I had enough. What next? How to make a poem out of sentences? I toyed with the idea of lineating them, turning them into verse, but then decided that prose was their

right form after all. Yet I didn't want one huge unreadable block. So I began looking for *joints* in this body.

These points of articulation give the whole thing a shape. They make it possible to say, for instance, that the poem has fourteen parts. (Does the fourteen-line shape of the sonnet have an influence here?) Even this simple division gives the text a useful kind of structure. A reader can see relations among 14 sections more easily than among 139 sentences.

Also, every break between sections is a place for both starting and stopping. If these two sentences are run together,

> Unless I had planned you, I would ask, How are the voices increasing? I am thinking this.

then "this" in the second sentence simply refers to the first sentence. But when the two are divided by a break between sections (II and III), "this" refers more to the whole poem. The break also emphasizes the link between the "you" in the first sentence and the explicit and implicit elements of address in previous sentences of section II. Addressing a "you" becomes part of what that section is *about*, part of its pattern of gestures.

One of the sentences worked equally well as a beginning point and an ending point. The one before it was a good ending, and the one after was another good beginning. The result was a one-sentence section:

X

> I was meaning to filter out meaning from the paper, but form—a heart—had charged something.

So isolated, the sentence becomes a comment on the project of the poem itself. It points to a kind of miracle in how a text begets an activity. In the first clause, "filter" and "paper" emphasize the mechanical operations of written language. "Paper" also suggests newspapers, the "white papers" of diplomats, and the dutiful "papers" of students, as well as the medium of print itself. "Meaning," repeated

this way, is reduced to mere willful busyness. (T. S. Eliot once said that "meaning" in poetry is the piece of meat that the burglar throws to the dog; it keeps the forebrain quiet while the poem does its real work.) The second clause answers with two metaphors: a "heart" that gives both physical and emotional life and the "charge" of what Whitman called "the body electric." These metaphors donate their force to "form," which once meant *beauty* or *arrangement* or the *wholeness* that makes anything a thing at all ("something"). "Form" is this section's word for how words rise up to become part of us. The sequence of verb tenses says that this essential activity of language was going on behind my back, even while I was trying to do something less.

It feels more accurate to say that I found these dividing points than to say that I created them. I discovered the poem's form within it. All that remained was to find a title. I wanted to emphasize what the poem was made of and the points where imagination was called on to fill the gaps of juxtaposition: the sentences and the boundaries between them. When I inventoried the poem, I found that it contained seventy-six assertions and sixty-three questions, and that's what I called it.