Regular Expressions as a System of Poetic Notation

by Dan Waber

"What i thot i was doing for the first 13 years was simply struggling with the question of how to notate my poems."—bpNichol, from "The "Pata of Letter Feet" in Meanwhile: The Critical Writings of bpNichol, Roy Miki, ed. (Talonbooks, 2002).

Words fail. Words fail even for poets, or, more accurately, especially for poets. The motivational force behind all acts of creation is dissatisfaction—I put forth the effort required to make this thing because I am sufficiently dissatisfied with a world in which it does not exist. I continue my attempts to make other, in the view of my early optimism, better things because I have grown dissatisfied with my earlier attempts, or, because I have found some new dissatisfaction I need to right, to write, two rate, too wrought. But I'm getting ahead of myself.

Words fail, especially for poets. Poets write poems because they are dissatisfied with a world in which the poem they are continually attempting to write does not exist. They have a poem in mind, a Platonic Idea of a poem that they can conceptualize completely, can hold in the mind intact, but which fails to survive translation from Idea to Manifestation with enough fidelity to create lasting satisfaction. So we write and we revise and we rewrite and we rerevise and we rerewrite and we rererevise, and we fail, continually and continuously.

We fail because words fail and words fail because we fail and so we begin to look at strategies for improving upon failure; not to break out of that loop (because all of life's great truths are paradoxes, are loops that encircle, that ensnare (though, of course, not all paradoxes are great truths)) but to refine the shape of the volume of the (metaphorical) space we scribe. We imagine that our poems are nets we cast into the onrush of experienced reality, nets that can catch actual truths if only we can fashion the knots and the loops in just the right way. That right way is informed by perceptions made from our individual points of view, thus our (individual) truth becomes a (generalized) truth when it is validated by the constructed net's ability to pull the same smelts of truth from the river of another's—an other's—experienced reality.

We begin where we are, in the one-dimensional, with the notion of words-as-tokens for concepts, and with the idea that the truth is escaping our nets, our poems, our language because of a problem of granularity. We need a finer meshed sieve. We read the words of more accomplished net makers, we become interested in etymology, connotation, denotation, language migration, we learn to select words for their finesse, for their resemblance to a scalpel rather than a maul. Now, instead of a net that can catch only points by accident, we can sometimes keep a segment of line-shaped truth in the net long enough to pull it near enough to the boat that we can watch the water flex as it rolls on its back and away. Then we start to make up our own words. We nail together things that have never been nailed together before, we break whole words down into morphemes and lexemes and phonemes and graphemes and soon we're up to our elbows in memes and faced with the task of tying together a net from a mountain of centimeter-long bits of string.

This takes space to work out, through, within. Compositional space. The page itself. Now we have both length and width. We have an open field, a blank slate, a way to build a net that is far more complex. Now we can tie knots in two dimensions instead of one, now we can catch things that have shape. Now we can see the fact of words as objects themselves, and not just referents to other objects. We can see language as a material with physical properties, with a reality all its own that is separate from, and able to act upon, its ability to make nets. Language is a thing not just a pointer to things. We are able to make this step from level to meta-level by taking a system which was fully developed for another purpose and applying that system to another set of conditions—not by pure innovation, but by appropriation. You don't need to relate to some particular movement of poetics to do this. If you understand the difference between:

```
no
   fair princess
```

and

```
no fair
        princess
```

you know all you need to know about what's been talked about so far.

This was still insufficient to the task. Next, we found ways to add time into the equation, made another meta jump, added another dimension to the complexity of the knots that could be tied in the net. We got hypertext, we got chance, we got the reader to assume responsibility for meaning-making, we got non-linearity, we got multi-linearity, we got all the entrances and exits into and out of the text that we could possibly want (again, it's worth noting (for what's coming next), not by innovation but by appropriation), but, we still don't have a satisfactory poem, still can't make a net that catches all the meaning we want to catch and compare. Yet.

The one thing all of these strategies have in common, though they represent a continuum that can be plotted as a range, is that they are anchored to the narrative structure that time (one way or another) inexorably imposes. This is the problem of narrative[i]. The problem can be solved, and it can be solved without innovation, with appropriation. Don't get me wrong, I'm not anti-innovation. All good things spring from the fount of innovation, initially. But innovation is idiosyncratic, inefficient, and often internally inconsistent, and these combine to make it bad at massive paradigm shift. Massive paradigm shift is much easier to accomplish with appropriation. Metaphor is a kind of appropriation. Metaphor doesn't attempt to innovate a completely new idea, metaphor appropriates a relation which already exists, applies it to a new system and says, "Just like this, only different."

I intend, in what follows, to make a case for poets and poetry to appropriate regular expressions (from mathematics and computer programming) as a system of notation to augment, to build upon, to multiply the possibilities of language to make nets for catching truth. This will not be a tutorial, a survey, or even a brief overview of regular expressions, as each of these is beyond the scope of this essay. For those interested in learning more about the rich, expressive, full set of possibilities I have included some references to further reading at the end. What I'm talking about doing is taking one language and multiplying it by another language. The complete details of the whole equation are left as an exercise for the reader. I will touch briefly on a few very specific aspects of how regular expression syntax works with the goal of shining a flashlight down a few of the streets and alleyways in what is really a metropolis of poetic potential.

A regular expression is a pattern used to match a pattern. By combining literal characters and meta-characters it becomes possible to describe not just specific strings of characters, but also whole classes of character string.

A pattern to match a pattern. At its simplest, `/pattern/` matches a p followed by an a followed by a t followed by a t followed by an e followed by an r followed by an n. The slashes delimit the beginning and end of the regular expression. Within the delimiters are the instructions as to what we want to match. In this example there are no meta-characters, only literal characters. Not very exciting, and not substantially different from what we do already with our current notation.

Meta-characters are special characters with a very specific syntax to govern their interactions. Meta-characters can be wildcards (which stand for any character), indicate alternation (this or that character or string of characters), capture matched sub-strings for later re-use (repetition, anaphora, structural distillation of forms), perform conditional operations (if this, then that), anchor a match to a portion of the line or a block of text, look ahead, look behind, limit other meta-characters, specify precise repetition, indicate levels of granularity (letter, line, word, arbitrary string), and, of course, more.

```
/sea?cret/
```

The question mark, in this context, means “possibly, but not necessarily, one of the previous character”. The pattern itself, then, is a literal s followed by a literal e followed by possibly, but not necessarily, one literal a followed by a literal c followed by a literal r followed by a literal e followed by a literal t. We’ve made the word secret conjure up a large body of water. In idiosyncratic systems that might be written “seacret” or “se(a)cret” or “se/a/cret” or a dozen other ways. None of those other ways is wrong. The issue here is that each way is author-specific so there’s a learning curve involved every time for the reader, and, it’s not likely to remain consistent through all that poet’s writing, nor to be an effective component in a larger system of notation that is able to leverage that consistency into combinatorial power. Individual poets have made individual solutions to individual problems on a case-by-case basis. Regular expressions provide a complete, consistent syntax for solving all possible notational problems.

```
/^th(is|at)(?!the other thing)/
```

This regular expression matches “this” or “that” (or, more accurately, it matches a literal t followed by a literal h followed by either (the choices are constrained by the parentheses) a literal i followed by a literal s or (indicated by the | meta-character) a literal a followed by a literal t) but only at the very beginning of a line (indicated by the ^ meta-character) and not (indicated by the ?! combination of meta-characters) followed by “the other thing”.

```
/\s+(\w+)\1\s+/
```

This regular expression contains zero literal characters, only meta-characters, and matches any amount of (but at least some) whitespace (tabs, spaces, or newlines, indicated by the `\s`), followed by any amount (but at least one) word character (letters, numbers, and a couple more characters, indicated by the `\w`) followed by that same string of found characters (indicated by the `\1` which references the contents of the first (leftmost) set of parentheses) followed by any amount (but at least some) more whitespace (indicated by the `\s`). Which means it matches *any* doubled words, like “this this” or “that that”.

In the first paragraph I said, “I need to right, to write, two rate, too wrought.” With regular expression syntax, I might write that as:

```
I need /t(w?o{1,2}) w?r(i|a|ough)te?/.
```

(literal t followed by an optional literal w followed by at least one but no more than two literal o followed by a literal space followed by an optional w followed by a literal r followed by (either a literal i, a literal a, or a literal o followed by a literal u followed by a literal g followed by a literal h) followed by a literal t followed by an optional literal e)

and then I get all the same things said, simultaneously (as preferred) rather than in an artificial order imposed by an inability to say more than one thing at the same time, plus, I get some other possible combinations that were previously unsaid (for example, the net of this particular regular expression will also catch “twoo rat”, which, if it were not desired, could be eliminated as a possibility by further refining the regular expression). By constructing poems with regular expressions what is really being built is a descriptor for a multiplicity of poems that exist in a state of potential.

To provide an illustration of the power of the combinatorial nature of the meta-characters in regular expressions, let’s take a look at three new ones (in addition to the `?` (possibly one) and the `|` (alternation) already explained.

`.` means one of anything.

`*` means zero to an unlimited amount of the previous character (or constrained grouping).

`+` means at least one, up to an unlimited amount of the previous character (or constrained grouping).

By combining these meta-characters we can obtain an astonishing degree of accuracy in describing what we want to mean. It is not an exaggeration to say that the range of constructions made possible by this ability to combine meta-characters into arbitrarily complex expressions is on par with mathematics.

`.*` means any amount of anything, including nothing.

`/ho+wl/` means a literal h followed by at least one, up to an unlimited amount of literal o followed by a literal w followed by a literal l. Thus, hoooooooooowl.

`/h(ow)+l/` means a literal h followed by at least one, up to an unlimited amount of (literal o followed by literal w) followed by a literal l. Thus, howowowowowl.

`/ho+?wl/` means a howl with at least one o, up to an unlimited amount of o, but constrained by the `?` to mean the least possible number of o rather than the default, which is the maximum number of o. The emotive implications of which, I believe, need no further elucidation.

Whitespace and self-commentary are also easily handled by regular expression syntax. Free-spacing mode is indicated by one of two methods. Either by placing an x after the closing / of the entire expression, or by enclosing a portion of the expression within a set of parentheses which opens `(?x` and closes `)`. Within free-spacing mode, all literal whitespace is ignored (spaces, tabs, and line breaks), though whitespace may still be included through other notations if needed. This can make complex expressions easier to untangle, or allow certain pieces to be broken apart for visual uses. Also, within whitespace mode, all characters between a # and a line break are ignored. This allows for any expression to include self-commentary. For example, we might take a line such as:

```
/sle[ea]p co-*mes too? (?:me){1,2}/
```

and put it into whitespace mode with:

```
/sle[ea]p co-*mes too? (?:me){1,2}/x
```

and then explain or offer other commentary on it like this:

```
/                   # not a bad place for a title.

sle[ea]p            # sleep as a kind of leap, a leap into
                    # sleep is a sleap is asleep.

co-*mes             # zero or up to an unlimited amount of
                    # dashes, which signal both a minus, and
                    # a conjoining, the possible range meant
                    # to invoke the uncertainty of when
                    # sleep will come

too?                # either to me or too me, in either case
                    # what will come to me is too me. maybe.

(?:me){1,2}         # me inside a pair of non-capturing
                    # parentheses which may occur once (me)
                    # or twice (making a meme), but must
                    # occur at least once (I think therefore
                    # I am), but no more than twice (no wail
                    # of mememe).

/x                  # end
```

Even with whitespace mode, the compactness and density of the notation can make it daunting to look at complex regular expressions at first. A poem like this may even look “ugly” when the meanings of the notations are not familiar:

```
/sle[ea]p co-*mes too? (?:me){1,2}, but in (?:un|re)fl?its
and  ?:(?:re)*(?:r?un)?)?(?:st)?art?s\. I sta(?:y|ge) up
(?:un)*til the (?:wh?e+ )+h?ours\. I
c?l(?:a|i)mb(?(?<=amb)or) my s?w(?:ay|eigh) (?:in)?to the
bed(?:room)?\. All?one, t?here is (k)?no(?(1)w)
goo?d(?:k?night’s sle[ea]p)?\./
```

However, the same factors which make regular expressions appear so daunting (their atomic nature, their combinatorial ease, their consistency, their brevity), the things which make them so powerful, are also the very same things which make their learning so manageable.

Already, I can write `/he(ck|ll)/` and you know that I mean heck and hell at the same time, and can appreciate the pun in the construction that makes heckle. Already I can write `/tim?e/` and you can apprehend in one glance the notion of being tied both by and to time. Already you can combine these two atoms of notation and take in the molecular multiple meanings of

```
/lip( s|s )t(i|a|u)cky?/
```

in a single glance (though perhaps only after you’ve worked it out once slowly). Already you’re thinking of the problems in your own poetry, already you’re wondering if they can all be solved by the right notation, already you doubt it, but already you’re willing to give it a try because already you’re dissatisfied with that poem you keep trying to `/w?ri(gh)?t?/`.

To learn more about regular expressions, I highly recommend Mastering Regular Expressions by Jeffrey Friedl (O’Reilly Media). Never in my life have I had the pleasure of reading a book that does a more thorough job of explaining a complex concept. If you want the kind of deep understanding that effective improvisation is based in, there is no better resource available. This is a textbook I re-read for pleasure because it’s so well done.

There are also a tremendous number of online resources for learning regular expressions. Most programming languages support some type of regular expressions though individual implementations may affect syntax. There are nuances and flavors in wide use that keep there from being One True Church of Regular Expressions, but, the overwhelming majority of basic concepts and their notations is consistent enough across these implementations for the purposes of appropriation by poetry. I spend most of my time with Perl regular expressions. An excellent online resource can be found at: http://perldoc.perl.org/perlre.html.

If you’re on a *nix system, you can simply type ‘perdoc perlre’ at a command prompt.

When reading about regular expressions substitute the word “means” wherever the word “matches” occurs and the text will become about poetics.

---

[i] “By placing words in sequence, narrative places events in sequence. By placing events in sequence, narrative describes a line in time. By describing a line in time, narrative ignores the tendency of each point (i.e. word) in the line to expand outwards in every direction, outwards forwards and outwards backwards, outwards upwards, outwards downwards and outwards outwards until each point becomes a hypersphere, an object whose center is everywhere and whose circumference is nowhere, which is another way of saying that the meaning emanating from each point in the line in time intersects and shares its space and every point in it with every other meaning emanating from every other point in the line in time, until there is no time and there is no space that is not occupied by every word in every sentence in every story that has ever been or ever will be told.”—from M, by Michael Aro, Starving Writers Publishing (2008).

This piece was originally published in August of 2008 in Vol. 5 of P-QUEUE (http://p-queue.org/)

Dan Waber is a Kingston, Pennsylvania poet, publisher, and multimedia artist. The hub of the online portion of his activities is logolalia.com.