Loop

Wilfried Hou Je Bek

The symbol of the snake nibbling away at its own tail, that mythological archetype of paradoxical repetition, is only partly suitable as a metaphor for the LOOP, that gargoyle of cyclical imagination in computation. The LOOP, a "reusable pattern where the language executes part of the pattern automatically, and you supply the parts that are different"[1] is one of the ways in which programming has gusto.[2] But it is not a single minded concept; the LOOP denotes a vast chain of beings (iterators, GO TO statements with passing arguments, count-controlled loops, condition-controlled loops, collection controlled loops, tail-end recursion, enumerators, continuations, generators, Lambda forms . . .) that crowd computer science and cloud the circumstances of its miracles.

Programming is an art[3] but we talk of computer *science*; this army of engineers has, however, failed to deliver us something like a looposcope, an instrument of vision that would augment our understanding of the manmade world we are trying to manipulate into constructs of unearthly beauty. This hypothetical apparatus would, in another medium, recreate with mnemogenic rhythm the striking experience of circularity produced by the straight-line forwardism of a discrete state machine.

The LOOP is an uphill continuum of abstractions. Some key moments stand out:

1. The humble origins of the LOOP when it leaks aboveground from the patterns carved on the stone that is the hardware.
2. The LOOP logically engineered for elegance on the slab of the programming language designer.
3. The release of the LOOP in the wild where typos, logical flaws, undecidability, and sloppy implementation haunt it. The LOOP needs only one opportunity in a run-time to become infinite.

In between these inauspicious moments, in the elephantiasis of abstraction and invention, in the syntactic sugar-coated manifestations of its form, the looposcope would be an invaluable aid to call its tail from its head and track

the movement of the LOOP through memory, its state permutations and its maneuvres in search space.

A read-write head lives along the infinite tape of a Turing Machine. Its behavior (the head moving back and forth, reading, writing, and erasing symbols after having been instructed to do so by symbols written on that tape) is the sum of all patterns created by the minimal instruction set that guides it. Imagine the following scenario: the head is instructed to JUMP to a certain position on the tape only to find when there another JUMP instruction telling the head to return to whence it came. There it will be instructed to go where it is now and so on and so on and so on, unconditionally and infinitely switching between states. Mostly a LOOP is merely a loop inside another. Traversing this control flow hierarchy the programmer climbs up and down, interrupting from above the loop that has become immortal below. The LOOP is a subset of all possible behavior made possible by JUMP (or BRANCH), the infinite loop is a special class within this set defined by the absence of interruption. The central position of the halting problem (the question of whether a computer given a certain input will halt, or run infinitely) in formal computation suffices to show that the LOOP is the foremost poetic entity in programming.

It is the goal of the programming language designer to provide powerful abstractions. For Alan Kay, designer of Smalltalk, these are "special ways of thinking about situations that in contrast with other knowledge and other ways of thinking critically boost our abilities to understand the world."[4] Such a statement succinctly aligns programming with the agenda of poetic theorists like Coleridge and Yeats. If we regard the loop as a species of tool for thinking about and dealing with problems of a certain nature, the sheer light-footedness of looping allows you to run away with the problem with more ease. Indeed, the debate over what constitutes the most elegant way to organize LOOPs from JUMPs is responsible for some of the most classic texts in computer science.

> If you look carefully you will find that surprisingly often a GO TO statement which looks back really is a concealed FOR statement. And you will be pleased to find how the clarity of the algorithm improves when you insert the FOR clause where it belongs

writes Peter Naur of the programming language Algol-60 in 1963, a comment quoted by Donald Knuth in his partly contemplative, mostly technical "Structured Programming with GO TO Statements."[5] Here Knuth traces the accumulation of resentment against GO TO statements that created the con-

ceptual agar on which Edsger Dijkstra's polemical "Go To Statement Considered Harmful,"[6] that grand diatribe against "spaghetti code," could proliferate with the success it did:

> For a number of years I have been familiar with the observation that the quality of programmers is a decreasing function of the density of GO TO statements in the programs they produce. More recently I discovered why the use of the GO TO statement has such disastrous effects, and I became convinced that the GO TO statement should be abolished from all "higher level" programming languages (i.e., everything except, perhaps, plain machine code).[7]

The hesitant Knuth, declaring his goal to be to help bring about the mythopoetic entity "Utopia 84," the first "really good programming language," fabulates moments of problem-solving agony when his mind, directed by the habit to use GO TO, was tied behind his back without it. Then he goes on to show how in certain cases a WHILE clause causes wasted cycles on the machine: the convenience of abstraction versus the responsibility of power. How do you find out if Element Y is present in Array X? The computer scientist has various ways to find out, throwing a zoo of loops at it and see what sticks best, but the ordinary webscripter just asks the interpreter "Is Y in X?" and the answer will roll out. Yet it is the LOOP that drives Miss Algorithm, the LOOP that sustains those creatures that live out in the sun. On the other hand, "Language is Fossil Poetry"[8] and who denies the schoolboy his moment of love made sedimental.

In every programming language higher than the hardware mimetic assembly language, the LOOP haphazardly diverges into two branches: iteration, in which "a collection of instructions [is] repeated in a looping manner"; and recursion, which has "each stage of the repetition executed as a subtask of the previous stage." Even though the two are often thought of as being "equivalent in terms of computer power"[9] they are radically different in the way they "feel" to programmers. In iteration, "shape is superinduced," while recursion is "form as proceeding" as Herbert Read said (about classical vs. romantic poetry).[10] Perhaps even Coleridge's famous distinction between fancy and imagination applies here (after all, Read was only paraphrasing Coleridge). In the Coleridgian view iteration would be "the imprisonment of the thing" and recursion the "self-affected sphere of agency."

The glossary in Programming PERL[11] offers definitions for both recursion and iteration. The length of each entry is telling. Iteration is merely, "Doing

something repeatedly." The entry for recursion begins: "The art of defining something in terms of itself," and ends: "[Recursion] often works out okay in computer programs if you're careful not to recurse forever, which is like an infinite loop with more spectacular failure modes." Recursion is surrounded in the programmer's mind with a nimbus of warm light captured in an oft-quoted bit of programmers' wisdom, variously attributed to L. Peter Deutsch and Robert Heller: "To iterate is human, to recurse, divine."[12]

Iteration branches off into two niche-driven subspecies canonized in most current programming languages as the primitives FOR and WHILE. Although often interchangeable, FOR is like a tourist that knows when it will be home (but with the power to RETURN earlier), WHILE is like a traveller away for as long as there is no hard reason to come back, potentially forever. Iteration requires special syntax, whereas recursion is the production of looping behavior generated by functions calling themselves. Iterations exist in a special time; recursion is behavior made up from the daily routines of life. Style, "that purification from insincerity, vanity, malignity, arrogance,"[13] is one reason for preferring one kind of LOOP, one instance of peripatetic know-how, above another. The nature of the memory to be manipulated, the way the magic carpet is folded[14] is another factor when deciding which LOOP to apply, which way to walk. Hash tables call for measures other than a one-dimensional list (Fibonacci numbers or a manifesto) or the nocturnal wandering through bi-directional structures (the world wide web or a city). Thinking in general and poetry in particular has forever been closely linked with iteration,[15] and was it not Coleridge who said that poetry is always a circuitous experience?

One aspect of the LOOP, and in many ways its defining quality, is the minimal means that result in automated infinite production. Is it when writing a simple FOR statement for the first time, counting to, say, 10 and printing to the screen at each iteration, that the novice programmer "Beheld the living traces and the sky-pointing proportions of the mighty Pan"?[16] This insight, its magic worn off in the mind of the experienced programmer as a mere fact of life, is that two simple lines of code can produce an "incantation" in which an effort as small as changing the upper limit increases the output to a "fairy-fountain" needing more time to be enacted than the computer it runs on will survive. Is it indeed not this raw force that allows permutation-sects to believe that the answer to the final riddles of the universe can be unwound by rephrasing them in a computational LOOP?

The LOOP is the powerhouse of worlds imagined in silico: the sweat-free producer of matter and time. It takes a Coleridge to do it justice.

Notes

1. Shriram Krishnamurti, *Programming Languages: Application and Interpretation*.

2. William Hazlitt, *The Spirit of the Age*.

3. Donald Knuth, *The Art of Computer Programming*.

4. Donald Knuth, Structured Programming with GO TO Statements.

5. A. C. Kay, "The Early History of SmallTalk," ACM SIGPLAN notices, Vol. 8, No. 3 (1993); available at http://gagne.homedns.org/~tgagne/contrib/EarlyHistoryST.html.

6. Edsger Dijkstra, "Go To Statement Considered Harmful."

7. Dijkstra, ibid.

8. Emerson, 'The Poet,' in Essays: Second Series, 1844.

9. J. Glenn Brookshear, *Computer Science*.

10. Herbert Read, *The True Voice of Feeling*.

11. Larry Wall, Tom Christiansen, Jon Orwant. *Programming Perl*.

12. James O. Coplien, "To Iterate is Human, to Recurse, Devine" in, C++ Report 10(7).

13. William Butler Yeats. *Synge and the Ireland of His Time*.

14. Vladimir Nabokov, *Speak Memory*.

15. See for instance, Rebecca Solnit, *Wanderlust: A History of Walking*.

16. Hazlitt, *The Spirit of the Age*.