Ishaan Jhaveri
May 15, 2017

Syntactically Speaking

The Euclidean form of proof begins with an axiom, a self-evident truth.

An atom is the smallest unit of matter that retains all of the chemical properties of an element.

A grammar is a series of rules that form the cornerstone of any language. These rules follow the Euclidean proof form in their structure. Here is an example of a grammar:

Critterish → Sentence**.** *
Sentence → Article Noun | Article Adjective Noun
Article → **the** | **a**
Noun → **tree** | **moon** | **river**
Adjective → **serene** | **perturbed** | **colorful**

In this grammar the language “Critterish” is composed of any infinite number of “Sentence[s]” followed by periods. That’s what the asterisk denotes. Each sentence in turn is composed either of “Article Noun” combinations or “Article Adjective Noun” combinations. “the” and “a” are the articles in this language, “tree”, “moon” and “river” the nouns and “serene”, “perturbed” and “colorful” the adjectives. Valid sentences in this (narrow but rather poetic) language would be “The river.”, “The serene moon.”, or “A perturbed tree.” Invalid sentences would be “A serene tree moon”, because nowhere in the structure are two “Noun[s]” allowed to occur together, “The yellow sun”, because “yellow” and “sun” are not in this language or “The perturbed river, a colorful moon”, because there is no comma in this language, sentences can only be separated by a period.

English (and Spanish, German, Mandarin, Hindi or any other language you can conceive) has a “Grammar” just like this, a more palatable (although in my humble biased opinion a considerably less precise) version of which was taught to most of us at a very young age, back when Physics, Chemistry and Biology were just called “Science” and Algebra and Geometry were indistinguishable from “Math”.

The first time I ever learnt about Grammars was in my freshman year of college, when for a class called “Data Structures and Object-Oriented Programming” I was building, programmatically, *Critterworld*. The inhabitants of *Critterworld* were simple creatures of habit. They occupied a world of hexagonal tiles. Each critter followed closely the “program” for the day that was randomly generated for it. The program was specified as a grammar that the critters knew how to parse. The grammar allowed them to eat, move or reproduce – and this was their world. All their desires were encapsulated perfectly into their language.

Formal Grammar like this is the ultimate symbol of my deepest passion. Mankind’s attempt to break down the nebulous vagaries of meaning into a relatable form that can be understood and traded in fascinates me. The audacity of such a project is beautiful. I see this project every time I sit down to write an essay, story or poem and every time I attempt, in Java, Python or C, to coax my computer into singing for me.

I have always felt that the burden of the software engineer is the burden of the poet. Both begin by identifying an idea, function or affect they are trying to create. For Keats when he sat down to write “Ode to a Nightingale” it was to meditate on the idea of mortality, begin a conversation about transience. The first program I ever wrote was called “isPrime” – it took a number as an input and outputted True or False depending on whether it was a prime number or not. The purpose of *Critterworld* was to simulate creatures that operated strictly according to a program, to teach the class about obvious and subtler aspects of code.

Fade far away, dissolve, and quite forget

What thou among the leaves hast never known,

The weariness, the fever, and the fret

Here, where men sit and hear each other groan;

Where palsy shakes a few, sad, last gray hairs,

Where youth grows pale, and spectre-thin, and dies;

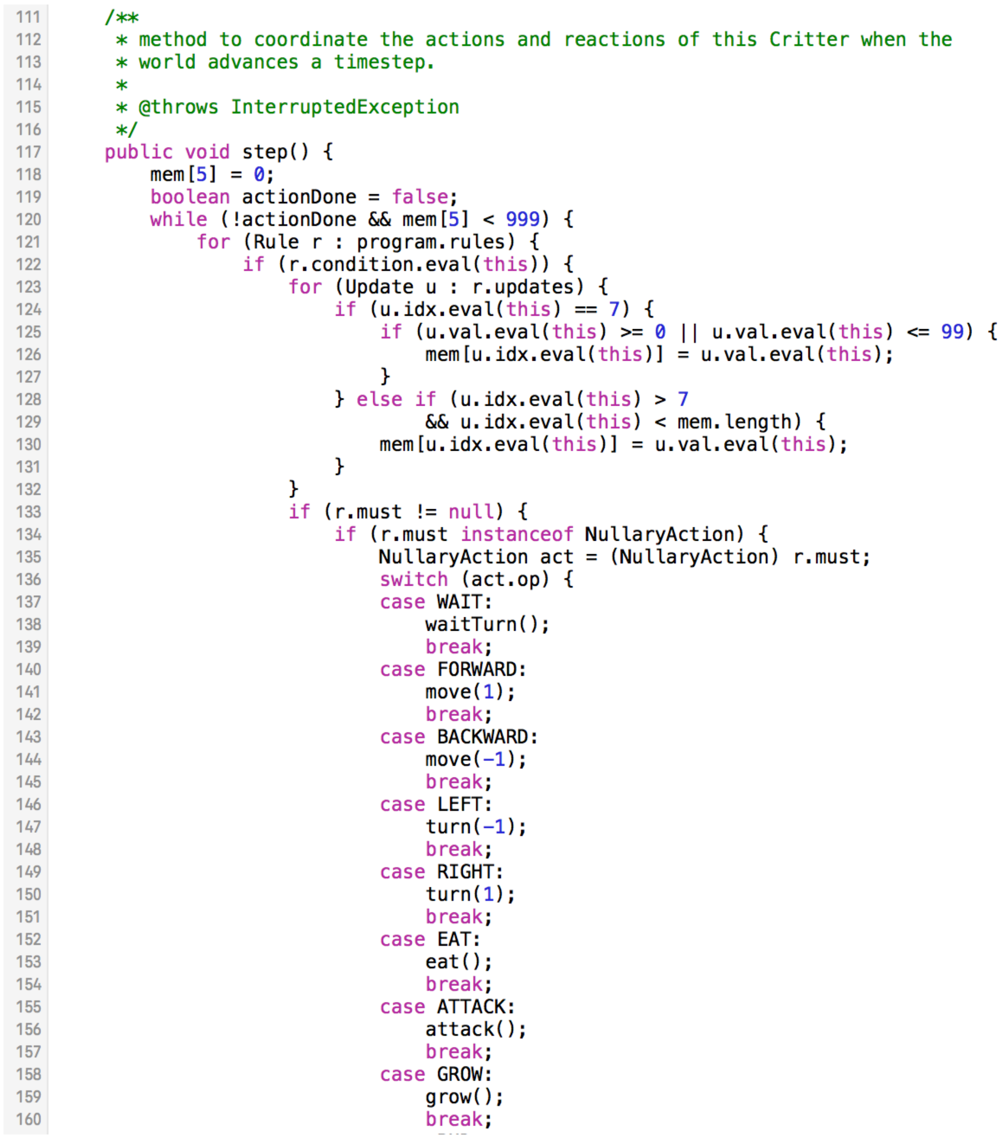
Where but to think is to be full of sorrow

And leaden-eyed despairs,

Where Beauty cannot keep her lustrous eyes,

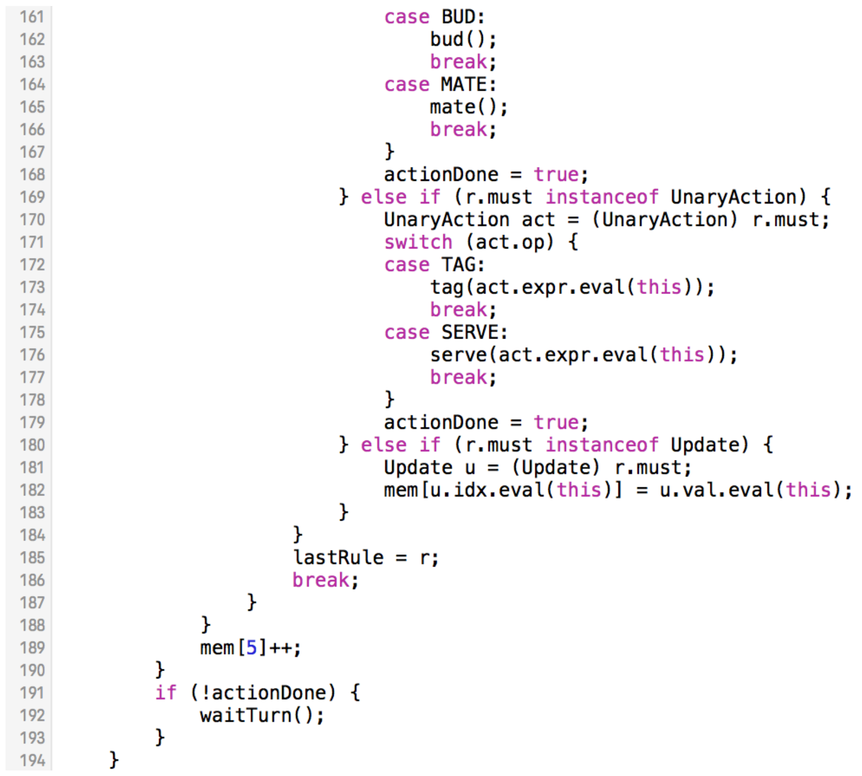
Or new Love pine at them beyond to-morrow.

Ishaan Jhaveri
May 15, 2017



```java
    /**
     * method to coordinate the actions and reactions of this Critter when the
     * world advances a timestep.
     *
     * @throws InterruptedException
     */
    public void step() {
        mem[5] = 0;
        boolean actionDone = false;
        while (!actionDone && mem[5] < 999) {
            for (Rule r : program.rules) {
                if (r.condition.eval(this)) {
                    for (Update u : r.updates) {
                        if (u.idx.eval(this) == 7) {
                            if (u.val.eval(this) >= 0 || u.val.eval(this) <= 99) {
                                mem[u.idx.eval(this)] = u.val.eval(this);
                            }
                        } else if (u.idx.eval(this) > 7
                                && u.idx.eval(this) < mem.length) {
                            mem[u.idx.eval(this)] = u.val.eval(this);
                        }
                    }
                    if (r.must != null) {
                        if (r.must instanceof NullaryAction) {
                            NullaryAction act = (NullaryAction) r.must;
                            switch (act.op) {
                            case WAIT:
                                waitTurn();
                                break;
                            case FORWARD:
                                move(1);
                                break;
                            case BACKWARD:
                                move(-1);
                                break;
                            case LEFT:
                                turn(-1);
                                break;
                            case RIGHT:
                                turn(1);
                                break;
                            case EAT:
                                eat();
                                break;
                            case ATTACK:
                                attack();
                                break;
                            case GROW:
                                grow();
                                break;
```

Ishaan Jhaveri
May 15, 2017



```
                            case BUD:
                                bud();
                                break;
                            case MATE:
                                mate();
                                break;
                            }
                            actionDone = true;
                        } else if (r.must instanceof UnaryAction) {
                            UnaryAction act = (UnaryAction) r.must;
                            switch (act.op) {
                            case TAG:
                                tag(act.expr.eval(this));
                                break;
                            case SERVE:
                                serve(act.expr.eval(this));
                                break;
                            }
                            actionDone = true;
                        } else if (r.must instanceof Update) {
                            Update u = (Update) r.must;
                            mem[u.idx.eval(this)] = u.val.eval(this);
                        }
                    }
                    lastRule = r;
                    break;
                }
            }
            mem[5]++;
        }
        if (!actionDone) {
            waitTurn();
        }
    }
```

Once the purpose of the poetic or programmatic project has been divined, the creator must choose a medium, a setting. For Keats this was through the parable of the nightingale, the “immortal Bird”, for me this was in Java using a “Objects” to represent critters. To Keats the nightingale exists in another world, ethereal yet concomitant with our own. In my program each critter is an “Object”, a well-defined segment of code with autonomous behavior. How each Object interacts with each other Object offers the insights into the nature of large programs that the assignment was designed to give us.

Now comes each painstaking, perfect, deliberate line. Lines of English that have been imbued with just the right amount of rhyme; and a rhythm that aides the poet’s greater goal. It was not lightly that Keats chose to portray the moon as a Queen, “Cluster’d around by all her starry Fays”. Each word has been calculatingly chosen. Each description’s bearing on the overall meaning taken into account. This Queen-moon helps evoke a sense of the real yet imaginary world the nightingale occupies. It lends context to many lines throughout the poem: “Charm’d magic casements, opening on the foam/ Of perilous seas, in faery lands forlorn.”

Lines of code that share the weight of the overall program’s function, that aim to be as clear and efficient as possible in their role as members of the larger project. On line 121 above “for (Rule r : program.rules)” teaches a simulated critter to loop through each “rule” in its daily program. Then, for each rule, it divines how to respond to it. In this single line, the entire inner logic of the program is driven forward. This line leads into other lines, the lines that take over when this segment of the program decrees the critter to “move” (line 144), “turn” (line 147), “eat” (line 153), “grow” (line 159).

The poet navigates symbols and motifs just as the programmer distributes information necessary to the program among different data structures. Keats’ discussion of “palsy” layers the conversation he is having much as my use of variables conveys to the program’s logic crucial information about a critter’s state.

Now comes editing/debugging. A test of the poem/program’s ability to fulfill its purpose. Finding a bug in a program, like finding a better phrasing for an idea, is like removing a stone from inside your shoe. There are always a few false alarms, you always remove stuff, think that its gone only for it to be back. Finally when you do isolate and remove the offending stone/line of buggy code/malapropism or misused idiom, it turns out to be an innocuous little thing. The poet hems and haws at the deepest recesses of his vocabulary, of his memory, fervently rephrasing and reorganizing till his point is made. Each symbol, sentence, stanza, transition can be put n different ways, each of which brings its own host of odd phrasing and awkward affect that need to be ironed out to preserve the sanctity of the greater product. The programmer too hacks at his code like

Heracles hacking at the hydra, but each minor fix brings in its place two further areas wanting attention. Getting at the root of what the code is supposed to express is finally the way forward, much like getting to the crux of an idea.

I sometimes wish I had a faithful proofreader to follow me around and proofread any snippet of anything I write, like I can with my code. I am constantly able to “test” my code, to confirm whether my meaning was understood by setting up outputs from my computer to tell me what it understood of my input, my meaning. I can’t do that with my writing quite so easily. Ultimately some programmers and poets will always wield their language better than others. Just as there are always those classmates of yours that write beautifully, those of us that code know that we have classmates/coworkers who code beautifully.

Unavoidably the poem and the program have an inner logic. A flow through which the lines and stanzas or the functions and if-else branches are organized and through which information meanders like a river into tributaries. This flow is crucial to readers of the poem’s understanding of it and for the compiler’s (the underlying system in a computer that reads and executes programs) understanding of the program. Ultimately it is this flow, composed of motifs, data structures, lines of English, lines of code, open to creative manipulation but conforming to some codified form, that delivers the ultimate force of either endeavor: the meaning. The meaning of a poem can evoke emotion in a reader, can inspire a reader to reconsider an aspect of their life, appreciate the beauty or suffering inherent in some topic the poet highlights. The language at this point has done its job, it has been the tongs through which the once ungraspable observations of the poet have been grasped by the reader. Thus form enables meaning. The meaning of the program can teach a machine to determine whether a seven-digit number is prime in under a second, can teach a computer to detect fraudulent credit card transactions, can teach a device planted in the ground of a farm near a sewage facility in rural India to determine whether the soil is too saline for the growth of wheat. Thus meaning can be conveyed across man and man, man and machine. Man has creativity, machine has power. Through language they can collaborate. Thus form enables meaning. Language enables meaning.

Fundamentally, English, Mathematics and the languages of Computer Science are just that - languages. They were constructed, constructed to fulfill some purposes and have since opened the possibility for others. Now from the compounding of these atomic units, comes about the creation of ever more constructs in these languages, applicable to more and more situations. This process of creating from combination and reorganization is to me the most enticing essence of creativity. To exist while these subjects develop via the continually more complex combination of their characters, founded of course on creation that came before, and to still have access to their historical firsts, be they grammar rules, axioms, or digital circuits, endows those that study them today with a certain familiarity of these atomic characters. The distinction of these subjects, that which ties Mathematics, English, and Computer Science together, is that those that study and practice it are responsible for that which their successors study.