AUG. 7, 2023, 3:32 P.M.

Raku: A Language for Gremlins

By Gremlins, For Gremlins™

Computer Things

I just added a big new section to learntla: Optimizing TLA+ Model Checking. I take a spec and then show 15 different optimizations, many of them getting a 10x runtime improvement. Patreon notes here. Besides that I've done very little writing the last couple weeks. I'm just in a slump: I sit down to write and the words all come out wrong. It happens sometimes and there's nothing to do about it but wait it out.

So instead of working I've been learning Raku.[1] It originally got on my radar after I ranted about dynamic languages and a couple of users told me I'd like Raku. I finally checked it out last week to see if it'd make a good "calculator language". I use a hodgepodge of Python, J, Frink, and Excel to do math and they all have their own big drawbacks, so it'd be nice if Raku could round them out.

After several days of experiments, I'm at a loss of *how* to describe Raku. The best I can come up with is that the language was designed by a bunch of really intelligent gremlins. Gremlins who spent a lot of time gathering feedback from other gremlins.

Weird Operators

Raku has no qualms about using Unicode operators. You check set membership with ∈. There's also ∉, ∋, and ∌. It's also fine with alphanumeric infix operators. String repetition op is x. Function composition is o.

```
> "a" x 3
aaa
> my &f = &sqrt o &abs
{noise}
> f -3
1.7320508075688772
```

`X` gives you the cross product of a list, `Xf` applies `f` to each element of the cross product, `Zf` does the same with zip.

```
> <a b c> X <1 2 3>
((a 1) (a 2) (a 3) (b 1) (b 2) (b 3) (c 1) (c 2) (c 3))
> <a b c> Xx <1 2 3>
(a aa aaa b bb bbb c cc ccc)
> <a b c> Zx <1 2 3>
(a bb ccc)
```

If `f` is an infix operator, then `[f]` reduces a list and `[\f]` accumulates it:

```
> [+] <1 2 3 4 5>
15
> [\+] <1 2 3 4 5>
(1 3 6 10 15)
```

~~ is the "anything goes" matcher.

```
> "abc" ~~ "abc" # do the strings match?
True
> "abc" ~~ Str # is abc a string?
```

```
True
> "abc" ~~ {.chars == 3} # is abc length 3?
True
> so "abc" ~~ /^b/ #does abc start with b?
False
```

Finally, there's `...`.

```
> 0,1,2...10
(0 1 2 3 4 5 6 7 8 9 10)

> 0,2,4...10
(0 2 4 6 8 10)

> 1,2,4...10
(1 2 4 8)
```

Defining operators

So you know how some languages let you define infix operators? Raku lets you also define new circumfix and *postcircumfix* operators.

```
# circumfix
sub circumfix:<[∀ zz>($inner){sum($inner)}
> [∀ 1,3,5...10 zz
25

# vector inner product!
sub postcircumfix:<| ⟩>(@left, @inside){[+] (@left Z* @inside)}
> <1 2 3>|<4 5 6>⟩
32
```

In addition to left- and right-associative infix operators, you can define operators to be chain associative (like how `x < y < z` is `x < y && y < z`) and "list" associative (`a op b op c` is `op(a, b, c)`).

Multiple dispatch

So Raku has "multiple dispatch", meaning that you overload a function with multiple different type signatures and it will choose the appropriate one.

```
# @ is for iterable types

multi f($x,@arr) {@arr.map(-> $elem {$elem + $x})}
multi f(@arr, $x) {@arr.map(-> $elem {$elem + $x})}
multi f(@arr1, @arr2) {@arr1 Z+ @arr2}

> f(2, <1 2 3>);
(3 4 5)

> f(<1 2 3>, <3 4 5>)
(4 6 8)
```

This isn't weird, lots of languages have multiple dispatch. What *is* weird is that you can also dispatch based on a *runtime predicate of the value*.

```
multi my_abs(Int $x where {$x > 0}) {$x}
multi my_abs(Int $x) {-$x}

> map &my_abs, <-4 4>
```

Also the signature of a function is a first-class value, as are the parameters in the signature.

Miscellaneous Things

- If you define a `MAIN` function, any parameters you give it will be automatically turned into CLI flags.
- Objects have way more preloaded methods than I've seen in any language, and I used to do Rails. The List object has methods for getting all permutations, all k-combinations, and all sliding windows.
- Junctions are this weird type-value-thing for doing multiple comparisons at once. 1|2 expands to `any(1, 2)`, so `1 < 1|2`. 1&2 expands to `all(1, 2)`, so `1 !< 1&2`.
    - Oh yeah and you can negate any infix operator by prefixing it with `!`.

- Raku is the only language I've ever seen that has `$kebab-case` names *and* infix subtraction, I'm guessing because sigils disambiguate `x-y`.
- The regex syntax isn't backwards compatible with Perl 5. For *thirty years* languages followed the PCRE "standard" and Perl 6 just… threw it all away.

That's just a tiny slice of all the weird Raku features, since I've only been looking at calculator applications so far. I haven't even learned about the object system, packages, or grammars! And even so, I *still* left out a ton of stuff. Like in a function body, `samewith` will call the same function with new arguments.

I think if I had to maintain a Raku legacy codebase my brain would explode. I have no idea how people would manage to write this language In The Large. At the same time, it seems incredible for programming In The Small. One-off scripts, computations, personal tooling, all the kinds of things I wanted to do with it in the first place.

I've also seen some really frustrating things:

1. The documentation is really poor and the heavy reliance on symbols makes it hard to search for things. I've learned a lot of poorly-documented languages but Raku is much bigger and more complex than any of them and it can be real demotivating.
2. The REPL crashes on Windows if I type in any Unicode. The compiler is also pretty slow, with even small files taking half a second or more. Iterating on things is painful.
3. I hate the sigil thing. I spent half an hour debugging a problem because I wrote `$x` instead of `@x`.

Overall? Maybe I'm just a gremlin at heart, but I think I like this language and want it to succeed. I assume I'll have a more balanced picture thoughts after I use it in anger. I just hope that over time the compile times and documentation improve.

---

1. Formerly known as Perl 6. ↩

If you're reading this on the web, you can subscribe here. Updates are 6x a month. My main website is here.