2024-05-02-Banning the Use of IF-THEN-ELSE

Banning the use of If-Then-Else

"If-then-else" has been one of the banes of our existence. The concept is too low-level. To get useful control flows, you have to tie *variables* into the equation and, then, you get into the issues of global variables, free variables and those sorts of things.

On the surface, it seems that "if-then-else" is extremely useful and cannot be a fundamental problem, because we've been indoctrinated to believe in the existence of if-then-else.

If-then-else was invented to implement conditional values of functions when using digital CPUs and subroutines. That's probably why McCarthy called the programming construct COND.

If-then-else was not originally meant to implement interesting control-flows and to abstract-away the use of GOTO.

We applied band-aids to our methods of programming CPUs, instead of stepping back and fixing the underlying problem by banning the use of low-level "if-then-else". This is like dispensing Tylenol® to dull pain, while not curing the cancer.

We have applied band-aids to the "problem" of control-flow in CPUs and subroutines. For example, we declare edicts such as not allowing globals, not allowing side-effects, etc. These edicts obviously contradict Reality. Servers and daemons, of course, have side effects, but our band-aids tell us that this cannot be possible. We become mentally paralyzed by cognitive dissonance. For example, programmers think that "concurrency is hard" only because our band-aids weren't designed to accommodate concurrency, yet, 5 year-old children learn hard real-time concurrency (piano lessons, reading music) without needing PhD degrees.

What can we do about this problem? How can we replace the use of if-then-else, while still achieving useful control flows? We've already seen small solutions to

the problem of if-then-else in function-based programming[1], e.g. in various *map()* functions. These are basically functional expressions of hoary bits of control flow that happen under-the-hood. We see ideas in FP creeping towards the goal with concepts like *pattern matching*.

With developments like OhmJS (based on PEG - parsing expression grammars), though, we can go whole-hog. We can invent textual syntaxes that express any control flow that we desire.

OhmJS is, itself, a shining example of convenient expression of a hoary kind of control flow. Simply looking at an OhmJS grammar reveals a control-flow that would be hard[2] to implement using if-then-else. OhmJS expresses a backtracking control-flow. “Try this branch, and, if it fails, backtrack and try the next branch...”.

---

[1] I consider function-based programming to be a superset of the current fad of FP-based languages. Function-based programming began in the early days of computing with languages like FORTRAN and Lisp. It was deemed convenient to use CPU subroutines to fake out mathematical functions. It appears to have been forgotten that the relationship is a one-way mapping only - functions can be represented using CPU subroutines, but, CPU subroutines are not functions.

[2] confusing

Appendix - See Also

See Also

References *https://guitarvydas.github.io/2024/01/06/References.html*
Blog *https://guitarvydas.github.io/*
Blog *https://publish.obsidian.md/programmingsimplicity*
Videos *https://www.youtube.com/@programmingsimplicity2980*
[see playlist "programming simplicity"]
Discord *https://discord.gg/Jjx62ypR (Everyone welcome to join)*
X *(Twitter) @paul_tarvydas*
More writing *(WIP): https://leanpub.com/u/paul-tarvydas*