2024-04-13-Notations For Programming

Choose Your Weapon

If you simply want to compute `x = sin(theta)` then you should use functional notation (incl. preemption) and TPLs (Textual Programming Languages).

But, if you want to do more with REMs (Reprogrammable Electronic Machines (aka "computers")), do *not* use functional notation and avoid the overhead of preemption (and associated TRAPs and extra management software, etc.)

What else might you want to do with REMs?

What might you want to do with a REM that doesn't involve building yet another calculator?

- Internet
- GUIs
- Robotics
- Blockchain
- Sequencing, iMovie
- etc.

Use both (or more) notations where appropriate

Function-based programming - which includes FP[1], but, is not restricted to FP - is a *simplifying assumption* that allows programmers to use mathematical notation mapped onto hardware.

There are other ways to use hardware. Other kinds of notations are more suited to those other kinds of uses.

Best of all worlds

In the best of all worlds, we would use *hybrid* notations that include functional notation *and* other notations that are better suited to problems-at-hand.

---

[1] FP means Functional Programming

An example might be blockchain. When viewed as a hybrid of technologies, blockchain is quite simple:

1. Cryptography (heavy math)

2. State-tracking protocols (simplistic state machines).

Blockchain tries to foil adversaries by chopping up cryptographic compute-ations into small, time-sequenced pieces, aka “states”. Attempts to express both parts of the hybrid technology using only a single notation - that of mathematics - leads to accidental complexity, especially in the state-sequencing code. When expressed in 2D, time-less mathematical notation, state-sequencing appears to be horribly complicated. When expressed as hand-drawn bubbles on a napkin in a diner, state-sequencing appears to be childishly simple.

The cryptographic aspects of blockchain require high-fallutin’ mathematics, but, the time-sequencing operations do not require high-fallutin’ mathematics.

Observation

Function-based notations are clunky when one wants to express control flow. At present, we use ad-hoc, unstructured forms of function-based notation that mis-uses COND combined with variables (aka "state").

This mis-use of function-based notation - stretching function-based notation beyond its sweet spot - leads to hoary bugs in code.

"State" is "bad" when you simply want to compute the value of an equation, but, "state" is not necessarily "bad" when you want to do other things.

Programmers have been indoctrinated into believing that function-based programming is the *only* good way to program REMs. I argue that this is not true.

Function-based programming is a convenient approach when you want to compute equations, i.e. using REMs as calculators. Function-based programming, though, is not so convenient when you want to do other things.

For example, observe the mess that "callbacks" produced. Observe the complexity of the concepts of “promises”, “futures”, “monads”, etc. that arise

when function-based programming is stretched beyond its sweet spot to describe actions that mathematical notation was not designed for, i.e. anything involving time, like servers, Flash, DAWs, control theory, etc.

The fact that you *can* express time-based concepts in textual, mathematical notation does not mean that you *should* use mathematical notation for these concepts. It may be convenient for mathematicians to manipulate these concepts on papyrus, but, that does not mean that this notation should be forced onto the act of programming hardware. In analogy, we know the atomic and molecular structure of plastics, but, factories that extrude plastics into practical end-user products don't directly deal with low-level concepts like the atomic and molecular structure of the plastics used in their products.

What is needed

We need research in ways of plumbing disparate notations - paradigms - together to produce programs for reprogrammable electronic machines.

We've been exposed to various attempts in these directions

- UNIX pipes
- networking protocols
- The admiration of Christopher Alexander's "Pattern Language" without the understanding of deep, underlying assumptions about fully isolated parts.

We've been over-emphasizing research into only a single notation, i.e. function-based programming that attempts to fake out mathematical notation on REMs.

We need more kinds of notations and more ways to join them together into final programs.

Somewhat ironically, we already hold in our hands the technologies that would allow us to completely isolate blocks of code - closures and FIFOs - but, due to our over-emphasis on function-based thinking, we diminish the full range of possibilities that come from these technologies.

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*