New-Breed HLLs

The way forward is: assume that ALL current programming languages are assemblers, and create new-breed HLLs that emit such assemblers.

Concurrency

Concurrency is a programming paradigm.

See Rob Pike's talk "Concurrency is not Paralellism" https://vimeo.com/49718712.

Parallelism is a specific problem. Parallelism and Concurrency are often conflated together.

Every parallel program employs the concurrent paradigm.

Not every concurrent program is parallel.

Concurrency is a programming paradigm that is useful on its own, like Structured Programming, OO, FP, etc., etc., etc.

Isolation

I invent the word "isolation" to mean uber-encapsulation.

Isolation encapsulates everything that encapsulation does, but, also encapsulates control-flow, whereas encapsulation does not.

Isolation exists in operating systems such as UNIX, but does not appear in commonly-available programming languages.

DaS

I invent the word "DaS" to mean Diagrams as Syntax.

It is possible to parse diagrams in the same way as we now parse textual languages.

The key ideas that I use are:

- Use backtracking (e.g. PROLOG) to do the parsing - we can optimize this later.
- Think of diagrams as *glyphs*, not pixels. For example, a box-and-arrow language contains only the *glyphs*:
    - boxes
    - arrows
    - text
- Allow glyphs to overlap, instead of insisting that they be arranged on a fixed, non-overlapping grid as is done with current programming editors.
- Mix text in with the diagrams.
- Some things are better expressed as text, e.g. $a = b + c$.
- Do not create diagrams to express things that we already know how to express.
- Use diagrams to express ideas that cannot be easily expressed[1] in text, e.g. composition of concurrent components, e.g. components that have multiple results, e.g. components that accept inputs at varying intervals of time, e.g. components that produce outputs at varying intervals of time.
- Use the Concurrent paradigm. DaS - e.g. box-and-arrow diagrams - tend not to work unless the boxes are concurrent. [Attempts at shoe-horning box-and-arrow diagrams into the sequential/synchronous paradigm cause difficulties and accidental complexities.]

[1] and understood

What Makes a Good Assembler?

"Good" assembler languages are ones that are easy to emit.

"Good" assembler languages are not the same as "good" programming languages.

Good assembler languges are "loosey goosey".

Good assembler languages do not require declaration-before-use.

Good assembler languages do not have strong-typing. Strong-typing is provided by the HLL, not the assembler. Strong-typing is for people, but assembler languages are for automation.

Good assembler languages provide 1st class functions.

Good assembler languages provide anonymous functions.

Examples of good assembler languages: Lisp, JavaScript.

Riffing

New-breed HLLs will riff on these ideas: concurrency, isolation, DaS.

Projectional Editors

Projectional editors edit assembler language.

A good place to start is with a "good" assembler language (see "What Makes a Good Assembler?").

SCLs

I invent the word "SCL" to mean Solution-Centric Language.

SCLs are a sub-class of DSLs[2], but are tuned to the details of a solution to **one** specific problem. In other words, what Engineers do[3].

PEGs

I believe the PEG[4] technology is a break-through technology that makes SCLs possible and allows them to be used liberally.

Ohm-JS

I am exploring Ohm-JS[5]. I have written about it and will continue to write about it.

[2] Domain Specific Languages

[3] Engineers solve one "real" problem at a time, using the best available technologies.

[4] https://bford.info/pub/lang/peg/

[5] https://github.com/harc/ohm