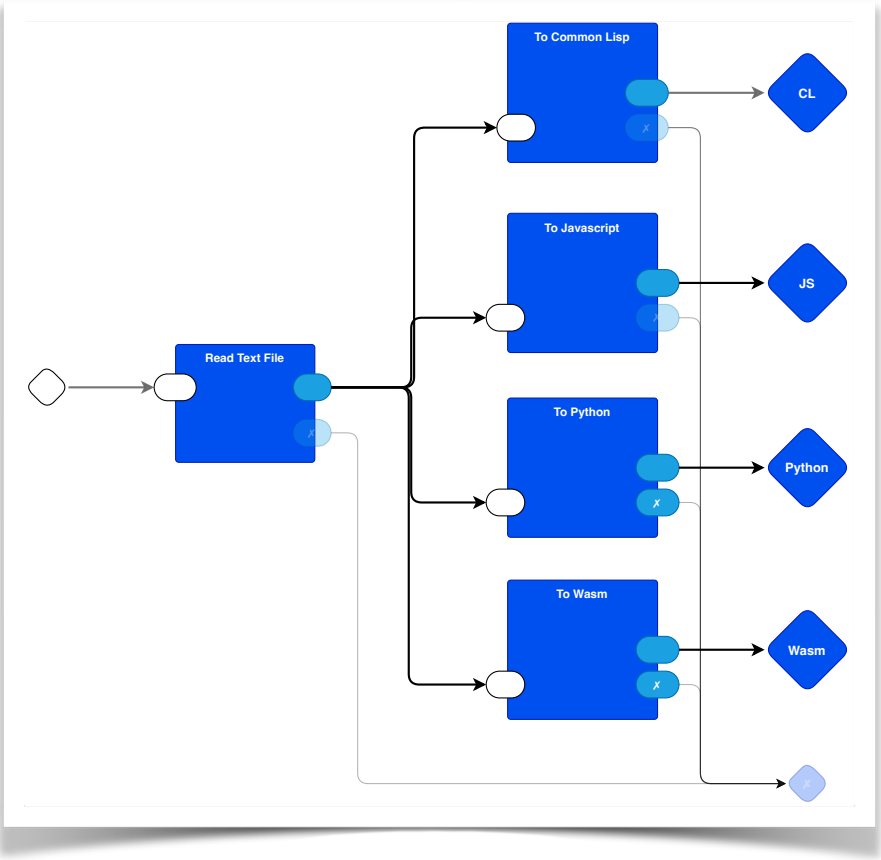
2024-04-04 IR - Intermediate Representations

IR - Intermediate Representations

With respect to the `arith0D` demo repository (Python POC: https://github.com/guitarvydas/0D/tree/devpy/python/test and Odin: https://github.com/guitarvydas/arith0d)...



The ultimate goal is to use Python/JS/CL/WASM as *assemblers* and never have to bother writing in such low-level 3GLs again.

The IR should be an HHL (higher-than-high-level language) that should allow easy machine manipulation and should allow easy transformation into other syntaxes.

Syntaxes are cheap, paradigms are hard.

Clojure is wildly more complicated than Lisp-without-macros. For example, in Lisp, you write recursive syntax using parentheses, in Clojure you need parentheses and square brackets (that's 100% more complicated :-).

The IDEs for Common Lisp are much more mature than the IDEs for any modern language, including Clojure. A significant portion of the goodness of Lisp has been lost due to the push for static compilation and type checking. The “dynamic” parts of Lisp, and the REPL, are good for iterative design refinement, whereas static compilation is good for Waterfall development of super-optimized code for Production Engineering activity. A REPL for a statically-typed-and-compiled language isn’t even close to being as useful as a REPL for a dynamically-typed language (or, even an untyped language, if you wish).

Ideally, the IR should allow you to work with any language and IDE, so it should just be a matter of choice and familiarity.

I haven't settled on anything yet, but, I admire Sector Lisp and OhmJS for their simplicity.

OhmJS makes it possible to use just about any syntax, not just that of Lisp.

The advantage of Lisp's "syntax" is that it allows recursive expression of syntax. I.E. Recursion at the syntactic level, not just at the runtime level. And, it is easy to parse using ad-hoc techniques (recursive descent) or tools like OhmJS and PEG.

Recursive syntax is especially easy to parse with OhmJS (and other PEGs) because PEG can parse matching brackets. CFGs, like YACC and stuff in the Dragon Book, make it cumbersome to parse that kind of stuff. REGEX is even worse - it lulls you into thinking that you can parse stuff with it, but, ultimately wastes your time when you try to scale up to more interesting grammars.

Additionally, in PEG, you can say "and skip this stuff as long as the brackets match", whereas in CFGs, simple ideas like that turn into mega-projects which tend to be avoided.

I think that Lisp has zero (0) syntax. Lisp is assembler with a recursive syntax, whereas Assembler is assembler with a line-oriented syntax (flat, non-recursive syntax).

In Lisp, you are forced to write programs by writing ASTs. Some people love this freedom, others hate it.

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*