T2T Transpiling

To Write A Compiler or Not To Write A Compiler?

I think that it's a waste of time to learn how to write compilers in the traditional manner, unless you really want to understand how that is done. LLVM seems to be the best way to write compilers "the old way".

I learned about compilers before LLVM existed, so I haven't bothered to track down books on LLVM. I would be surprised if The Dragon Book is still a good resource relative to newer offerings.

On the other hand, if all that you want to do is to write a new language, I strongly advise using a text-to-text technology based on PEG parsing ideas. Don't write a compiler - we have lots of those already. Just write your new language and make it produce code for one of the existing compilers, say, new-language->JavaScript. JavaScript is OK, as long as you don't have to use its syntax. Let a machine write Javascript for you.

Using PEG technology instead of CFG-based technology, you can crank out something in about an afternoon. I guess you'd need an additional several days to come down the learning curve. I don't remember ever being able to build a CFG-based compiler/interpreter/transpiler in only a few hours nor even a few days.

My favourite PEG technology is a tool called OhmJS. It comes with a REPL for grammar-building called ohm-editor.

PEG is just an old technique cleaned up and formalized. It used to be called "recursive descent". PEG adds backtracking to this technique to make it wildly easier to use.

The best pre-OhmJS technologies that I know of for writing MVIs[1] of new languages are TXL and Lisp. TXL was/is taught at Queen's University in Kingston,

[1] MVI - Minimum Viable Implementation. Scrimp on efficiency instead of the product design, as is suggested by MVP.

Ontario, Canada, and has been around a *long* time. TXL was using the buzz-word "functional" long before "functional" became a buzz-word.

It *looks* like you could do a lot of this by just using REGEXs, but, if you try that, you will end up in the weeds when you try to scale to anything more than a simple line-based pattern match. Most popular programming languages are not line-based, but use multi-line, structured text. PEG is better than REGEX and PEG is easier than CFG. I would say to learn OhmJS[2], and, to only learn about CFGs and REGEX and Parser Combinators if you are into self-flagellation.

[2] or other PEG, but, Ohm is better than all the rest of the PEGs I've seen

In a 'real' Computer Science, the best languages of an era should serve as 'assembly code' for the next generation of expression.

Alan Kay on youtube - see 31:50 [3] https://www.youtube.com/watch?v=fhOHn9TClXY&t=859s

[3] Thanks to Rajiv Abraham for sending me this clip.

Old way: write Yet Another Compiler.

New way: cheat, just write a grammar, then, convert incoming source code to an already-existing language.

The “new way” isn’t that new, it’s simply been overlooked in the efforts to over-complicate everything.

Step 1: learn how to write grammars.

Step 2: learn OhmJS[4].

Step 3: write a t2t[5] transpiler.

Step 4: Stop here.

[4] or other PEG, like ESRAP, peg.js, etc.

[5] text-to-text

A “compiler” isn’t all that special. It’s just a *big* program which has to work right.

A “compiler” is just a t2t transpiler itself. It transpiles “high level” source code into “low level” assembler code.

Why is it easier to write *t2t*s in 2024 than it was in the 1950s?

PEG.

In the 1950s you had to hand-craft a recursive-descent parser, whereas in 2024 you can use Ohm to help you do this. In the 1950s, you had to learn about language theory and CFGs and LR(k)s. That stuff is about *language* design, not *parser* design. To build a full-blown, robust language, you will likely need to deal with *language theory*, but, to fool around with creating a new language iteratively, you don’t need all that stuff. Defer those details until you are satisfied with your design, or defer until never.

I argue that deep-diving into over-complicated technologies, like CFGs, has caused us to miss the obvious, low-hanging fruit, like parsing diagrams, instead of only parsing text. In fact, I view all of Bret Victor’s Worry-Dream stuff as simple pet tricks:

(1) Use syntax that goes beyond text, i.e. simple diagrams and glyphs and graphics, and,

(2) Provide a REPL. Moving a slider to make a ball go back and forth is no different from eval-ing a subroutine on the command line, except that the former uses modern hardware.

Given XML, or SVG, and Ohm, it’s easy to parse diagrams. If you bolt a t2t onto that workflow, you get a diagram-to-Javascript (or WASM, or LLVM, or Lisp, or Python, or Haskell, or ...) transpiler.

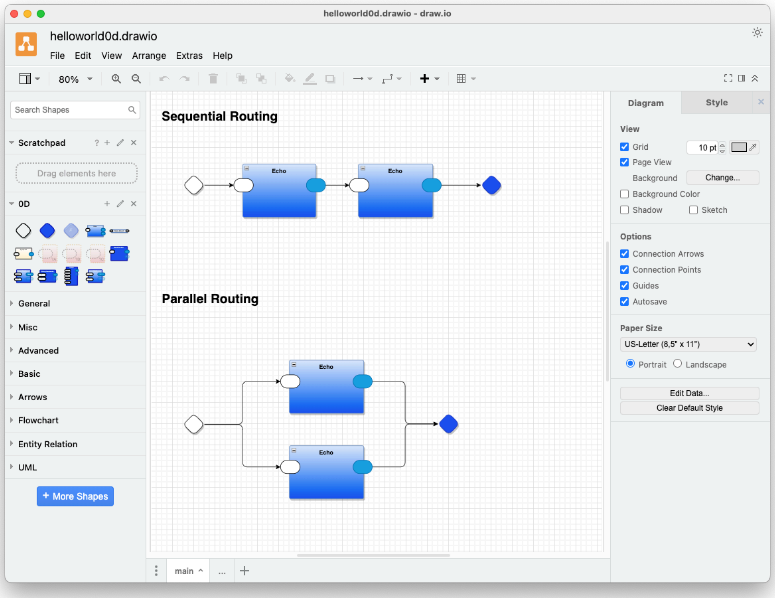
If anyone is interested, I provide links to a sourdough starter and some simple examples that are in my repo.

Warning: it’s actually disappointing and underwhelming. It’s not over-complicated enough.

Disclaimer: although I've been using these ideas for decades, I'm still playing around with the best way to package the ideas using existing, text-based tools. YMMV.

Examples

Hello World



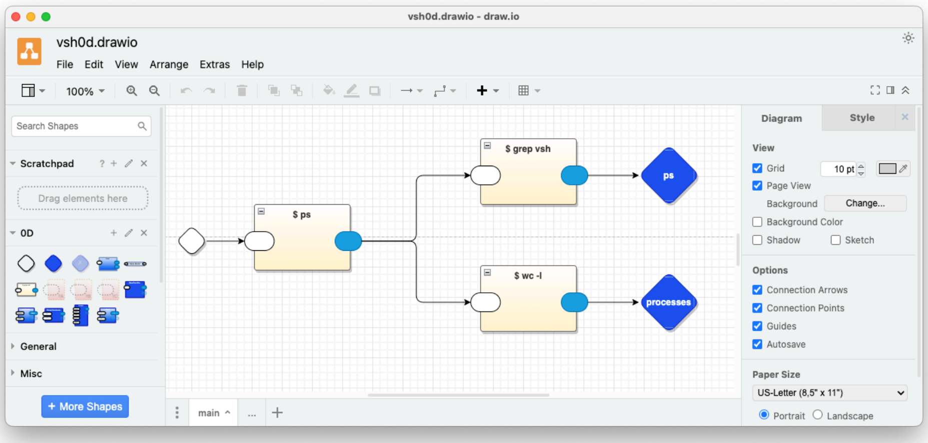
View
☑ Grid 10 pt
☑ Page View
Background Change...
☐ Background Color
☐ Shadow ☐ Sketch

Options
☑ Connection Arrows
☑ Connection Points
☑ Guides
☑ Autosave

Paper Size
US-Letter (8,5" x 11")
[✓ Portrait] Landscape

https://github.com/guitarvydas/helloworld0d

VSH (Visual SHell)



View

- [✓] Grid 10 pt
- [✓] Page View
- Background Change...
- [ ] Background Color
- [ ] Shadow [ ] Sketch

Options

- [✓] Connection Arrows
- [✓] Connection Points
- [✓] Guides
- [✓] Autosave

Paper Size

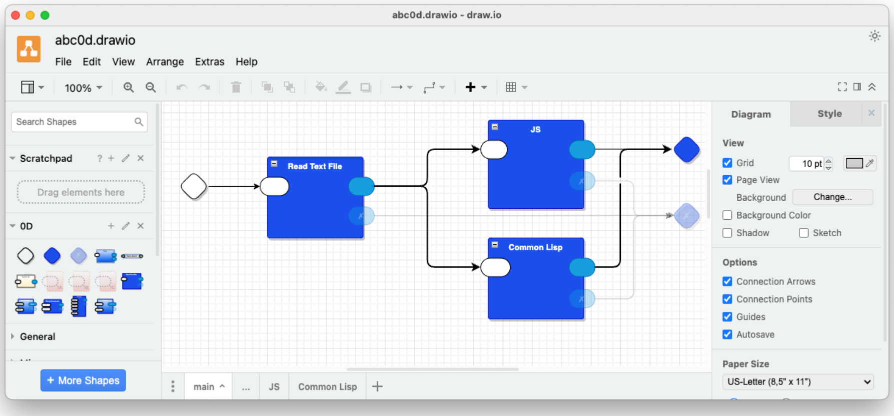
US-Letter (8,5" x 11")

[✓ Portrait] Landscape

https://github.com/guitarvydas/vsh0d

ABC

Trivial example language that t2t transpiles to Javascript and Common Lisp



☑ Grid 10 pt
☑ Page View
Background Change...
☐ Background Color
☐ Shadow ☐ Sketch

☑ Connection Arrows
☑ Connection Points
☑ Guides
☑ Autosave

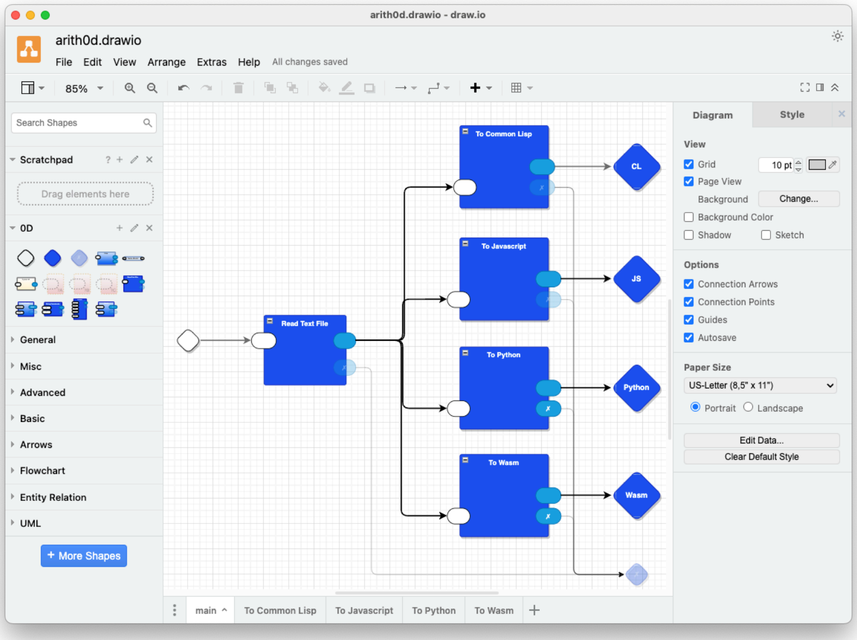
US-Letter (8,5" x 11")

https://github.com/guitarvydas/abc0d

Arith

OhmJS' arithmetic example that t2t transpiles to WASM, Python, Javascript and Common Lisp

https://github.com/guitarvydas/arith0d



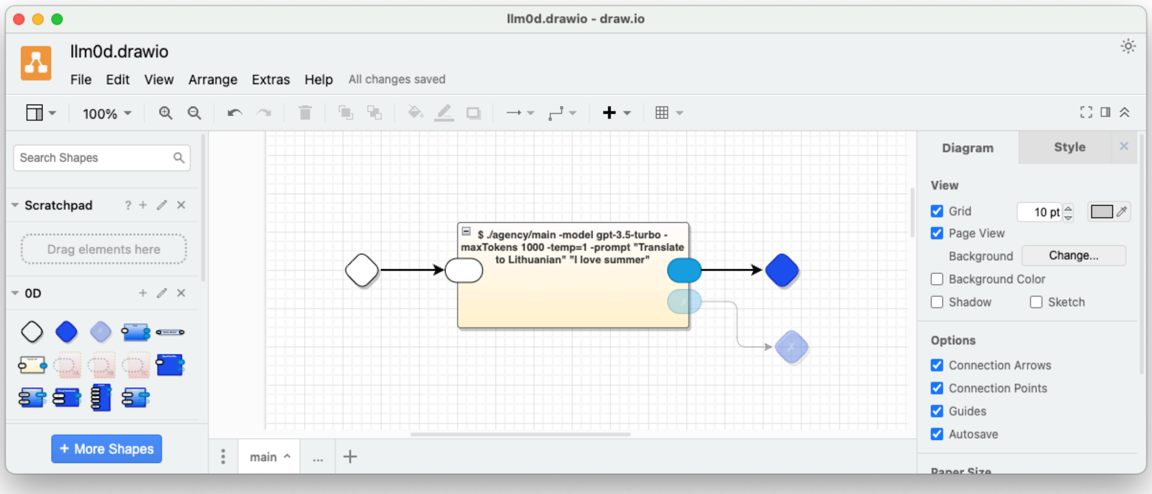
View
☑ Grid 10 pt
☑ Page View
Background Change...
☐ Background Color
☐ Shadow ☐ Sketch

Options
☑ Connection Arrows
☑ Connection Points
☑ Guides
☑ Autosave

Paper Size
US-Letter (8,5" x 11")
● Portrait ○ Landscape

LLM

Simple use of an LLM (N.B. Steve Phillips is exploring how to use an LLM to do t2t transpilation for creating a new programming language. This `llm0d` example is, maybe, a stepping stone for that kind of work.



```
$ ./agency/main -model gpt-3.5-turbo -maxTokens 1000 -temp=1 -prompt "Translate to Lithuanian" "I love summer"
```

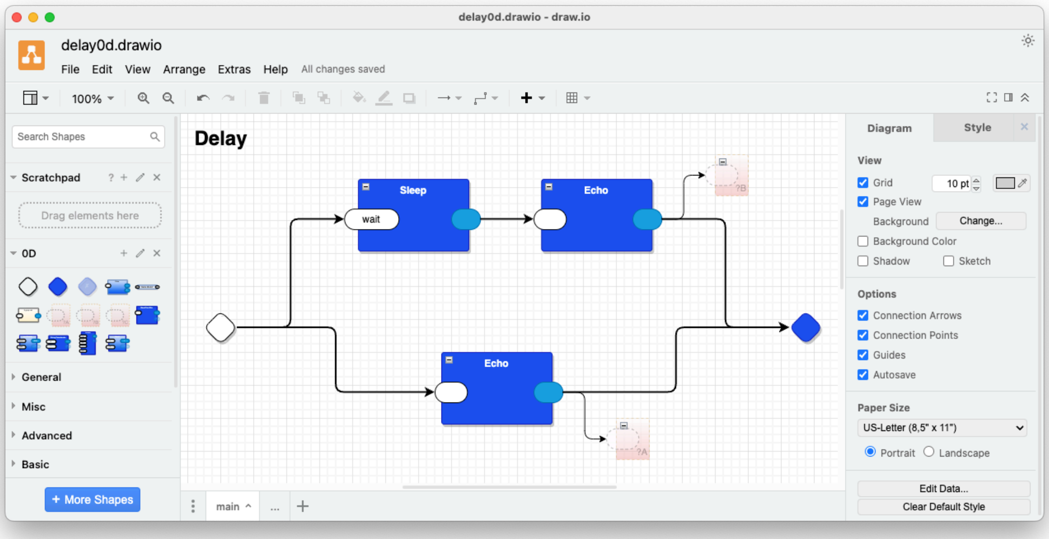
Grid ✓ 10 pt
Page View ✓
Background Change...
Background Color ☐
Shadow ☐
Sketch ☐

Options
Connection Arrows ✓
Connection Points ✓
Guides ✓
Autosave ✓

https://github.com/guitarvydas/llm0d

Delay

Example of how to deal with long-running processes, and, example of using probes.



View
✓ Grid 10 pt
✓ Page View
Background
☐ Background Color
☐ Shadow ☐ Sketch

Options
✓ Connection Arrows
✓ Connection Points
✓ Guides
✓ Autosave

Paper Size
US-Letter (8,5" x 11")
[✓ Portrait] Landscape

https://github.com/guitarvydas/delay0d

Starter

https://github.com/guitarvydas/0D

See `README.md` for instructions.

WIP

Projects I've started, but mostly out of laziness, haven't completed. I would be glad to explain, help and kibitz if anyone else wants to dabble or to pick up and continue.

Scheme to Javascript transpiler, diary in https://guitarvydas.github.io/2020/12/09/OhmInSmallSteps.html .

Kinopio to markdown converter, then, to LLM, first cut in https://github.com/guitarvydas/kinopio2md .

Markdown as a programming language syntax, prototype https://guitarvydas.github.io/2023/09/24/Find-and-Replace-SCN.html .

Macros for non-lisp languages, WIP in https://guitarvydas.github.io/2024/01/05/Macros-for-Non-Lisp-Languages.html .

PT Pascal compiler upgraded for 2024, WIP in https://github.com/guitarvydas/ptpascal0d .

Dungeon Crawler game inspired by Ceptre, slides in https://github.com/guitarvydas/ceptre/blob/jan17/presentation/Ceptre%20Walkthrough.pdf , WIP in https://github.com/guitarvydas/ceptre/tree/jan17/dc0D (view with drawio) branch "jan17".

CL0D - rewrite of the 0D engine in a more recursive form, hoping to follow up by "lifting" the rewrite to a meta syntax and using a t2t to create engines in Odin, Python, Javascript, CL, etc. branch devcl0d https://github.com/guitarvydas/0D/tree/devcl0d .

Appendix - See Also

References

https://guitarvydas.github.io/2024/01/06/References.html

Blogs

https://guitarvydas.github.io/

https://publish.obsidian.md/programmingsimplicity (see blogs that begin with a date 202x-xx-xx-)

Videos

https://www.youtube.com/@programmingsimplicity2980

Books

leanpub'ed (disclaimer: leanpub encourages publishing books before they are finalized - these books are WIPs)

https://leanpub.com/u/paul-tarvydas

Discord

https://discord.gg/Jjx62ypR

all welcome, I invite more discussion of these topics, esp. regarding Drawware and 0D

Twitter

@paul_tarvydas

Mastodon

(tbd, advice needed re. most appropriate server(s))