

2024-05-05-Free Range Programming

The Power of PEG	2
Appendix - See Also	4

The Power of PEG

PEG - and my favourite OhmJS - can match patterns in text in ways that CFGs can't.

To demonstrate, here's a snippet of code. Don't worry about what the code says, just notice that it begins with

```
def Das2json
```

And ends with

```
return _r.return_string_pop ()
```

I don't care about all of the stuff in between.

```
def Das2json (_r):
    _r.push_new_string ()
    _r.begin_breadcrumb ("Das2json")
    XML (_r)
    _r.append_returned_string ()
    Spaces (_r)
    _r.append_returned_string ()
    _r.need (_r.endchar ())
    _r.end_breadcrumb ("Das2json")
    return _r.return_string_pop ()
```

Using a CFG-based parser, I would need to write out, in detail, a grammar for all of the stuff in between the beginning and the ending phrases.

With OhmJS, though, I can skip over the stuff in the middle and just match for the beginning and the ending phrases.

Below is an experimental OhmJS grammar that matches the beginning and ending phrases without making me write a grammar for all of the stuff in between. It's kinda like REGEX, only more powerful:

```
defname {
  defName = "def" spaces name spaces through<"return _r.return_string_pop ()">
  through<s> = (~s any)+ s
  name = letter alnum* ~alnum
}
```

The screenshot shows the Ohm Editor web interface. The left pane displays a grammar definition:

```

defname {
  defName = "def" spaces name spaces through<"return _r.return_st
  through<s> = (~s any)+ s
  name = letter alnum* ~alnum
}

alnum (an alpha-numeric character) = letter
  | digit

any (any character) = /* primitive rule */

letter (a letter) = lower
  | upper
  | unicodeLtm0

spaces = space*

```

The right pane shows the parse tree for the string "D a s 2 i s o n (". The root node is "def", which has children "spaces", "name", and "spaces". The "name" node further expands into "letter", "alnum*", and "spaces". The "letter" node expands into "upper", "lower", "digit", "letter", "letter", "letter", "letter". The "alnum*" node expands into "alnum", "alnum", "alnum", "alnum", "alnum", "alnum", "alnum", "alnum". The "spaces" node expands into "any", "any", "any".

An "Edit example" dialog is open, showing the start rule "defname > (default)" and the following code:

```

def Das2json (_r):
  _r.push_new_string ()
  _r.begin_breadcrumb ("Das2json")
  XML (_r)
  _r.append_returned_string ()
  Spaces (_r)
  _r.append_returned_string ()
  _r.need (_r.endchar ())
  _r.end_breadcrumb ("Das2json")
  return _r.return_string_pop ()

```

At the bottom right, there are checkboxes for "Explain parse" and "Show spaces", and a set of navigation arrows.

Would I use this in production code? No.

Would I use this in development code? Yes.

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>