pigeon - Generate parsers in Go from PEG grammar

Bärner Go Meetup – 27.06.2023

Agenda

1. What is Parsing Expression Grammar (PEG)
2. What is pigeon
3. Examples
4. Wrap-up

Parsing Expression Grammar (PEG)

History & Properties of Parsing Expression Grammar

Initial proposal and implementation by Bryan Ford at MIT in 2004.

Goal: overcome limitations of other grammars (e.g. CFG) and regular expressions.

Packrat parser: top-down parsing with backtracking, unlimited lookahead and linear parsing time with memoization.

PEG gained attention among researchers and developers, which lead to a variety of implementation in JavaScript, Python, Ruby, Java and obviously Go.

What is “Parsing Expression Grammar” (PEG)

- Grammar which allows to describe a *formal language*
- Consists of
  - a set of nonterminal and terminal symbols
  - a set of rules to recognize strings in a language
  - a starting rule (or expression)
- A rule is an *identifier* followed by a *rule definition operator* and an *expression,* e.g. `identifier = expression`
- Examples of rule definition operators: `=`, `<-`, ← (U+2190), ⟵ (U+27F5)

Example:

```
RuleA = 'a'+
```

Demo

https://pegjs.org/online

PEG nonterminal and terminal symbols

sequence: `a b`

ordered choice: `a / b`

zero or more: `a*` , one or more: `a+` , optional: `a?`

followed by: `&a` , not followed by: `!a`

nonterminal symbols

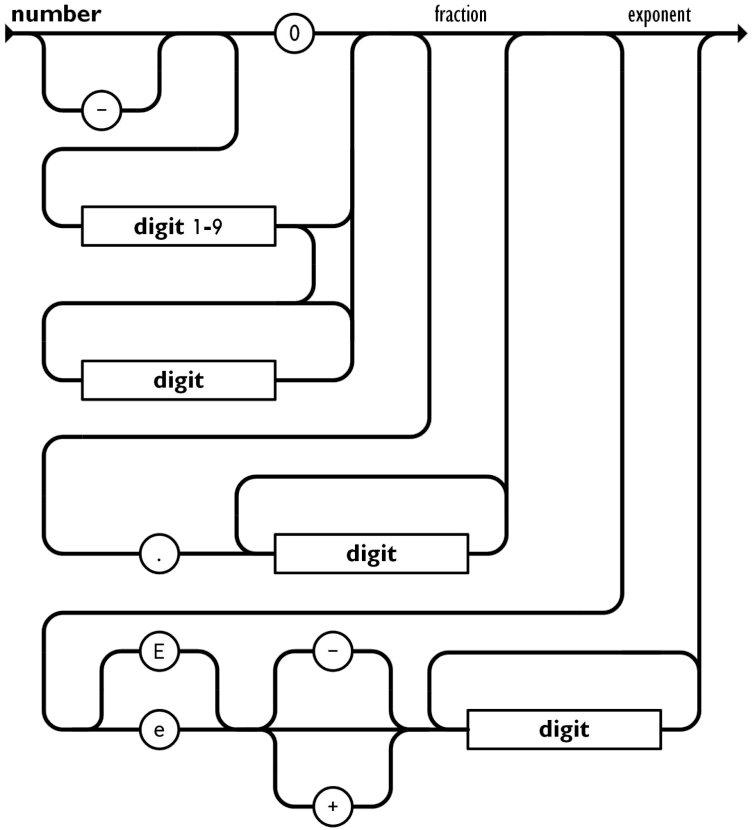
any character: `.`

single char literal: `'x'` , string literal: `"abc"`

character class: `[ab]` , character range: `[a-z]`

terminal symbols

Parsing of a JSON Number

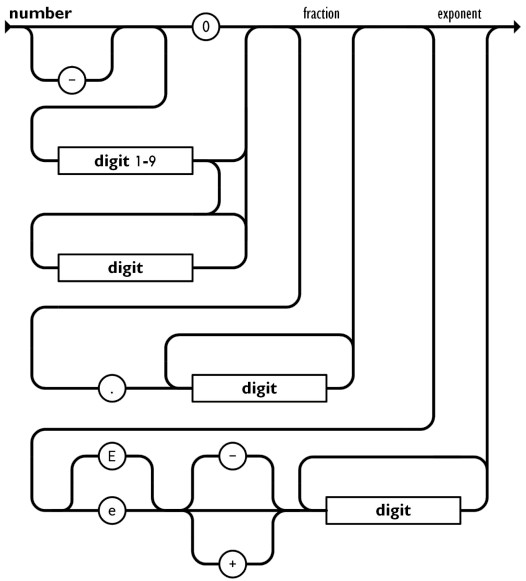


Source: https://www.json.org/

Parsing of a JSON Number (cont.)

```
Number = '-'? Integer Fraction? Exponent?
Integer = '0' / NonZeroDecimalDigit DecimalDigit*
Fraction = '.' DecimalDigit+
Exponent = 'e'i [+-]? DecimalDigit+

NonZeroDecimalDigit = [1-9]
DecimalDigit = [0-9]
```



Parsing of a JSON Number (cont.)

```
Number = '-'? Integer Fraction? Exponent?
Integer = '0' / NonZeroDecimalDigit DecimalDigit*
Fraction = '.' DecimalDigit+
Exponent = 'e'i [+-]? DecimalDigit+

NonZeroDecimalDigit = [1-9]
DecimalDigit = [0-9]
```

sequence
ordered choice
zero or more
one or more
optional
single char literal
character class
character range

Demo

https://pegjs.org/online

Left Recursion

- PEG is *well-formed* if it contains no ***left-recursive rules***

Not allowed:

```
string-of-a = string-of-a 'a' | 'a'
```

Rewritten:

```
string-of-a = 'a'+
```

Left Recursion cont.

Not allowed (grammar tries to express precedence order or products)**:**

```
Expr    = Product / Sum / Value
Product = Expr (('*' / '/') Expr)*
Sum     = Expr (('+' / '-') Expr)*
Value   = [0-9]+ / '(' Expr ')'
```

Instead (precedence order is inverted)**:**

```
Expr   = Term (('+' / '-') Term )*
Term   = Factor (('*' / '/') Factor )*
Factor = [0-9]+ / '(' Expr ')'
```

pigeon

What is pigeon

- Command written in Go
- Generate parsers in Go from parsing expression grammars (PEG)
- Inspired by PEG.js project

- Support for memoization and grammar optimization (experimental)
- Extensions to standard PEG (e.g. state, state change code blocks, failure labels, throw and recover, global store, support left recursion might be added in PR #123)

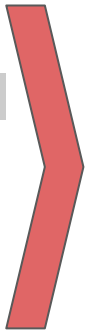
- Original implementation by Martin Angers (github.com/mna)
  Maintained by myself (github.com/breml) since 2017

Link between PEG and Go code

Small Example (Count Words):

```
{
  package parser
}

wordCount = w:(word delim?)* EOF {
  words := toAnySlice(w)
  return len(words), nil
}
word = letter+
delim = ' '
letter = [a-zA-Z]
EOF = !.
```



```
package parser


func (c *current) onwordCount1(w
interface{}) (interface{}, error)  {
    words := toAnySlice(w)
    return len(words), nil
}
```

toAnySlice / toIfaceSlice - An Important Helper

- pigeon operates on `interface{}`, which recently became `any`
- Some of the PEG expressions return a list of values (sequence, `*`, `+`)
- `toAnySlice` converts an `any` value to a slice of `any` values (`[]any`)

```
func toAnySlice(v any) []any {
    if v == nil {
        return nil
    }
    switch v1 := v.(type) {
    case []any:
        return v1
    case any:
        return []any{v1}
    Default:
        return nil
    }
}
```

Demo: wordcount

Exercises

- Allow newlines.
- Allow punctuation marks (which are often follow by a space).
- Allow umlauts.

Of course we can use a PEG parser also in a command.

Examples

Demo: CSV parser

What can it be used for - more involved use-cases

- Tichu Log Analyzer
  http://tichulog.brettspielwelt.de/

- Logstash Config
  https://www.elastic.co/guide/en/logstash/current/config-examples.html
  https://github.com/newrelic/logstash-examples/blob/master/mutate.conf
  https://github.com/breml/logstash-config/blob/master/logstash_config.peg

Logstash

- Asciidoc library
  https://github.com/bytesparadise/libasciidoc/blob/master/pkg/parser/parser.peg

Is it fast?

Compare PEG JSON parser with stdlib:

```
$ go test
-bench='^(BenchmarkPigeonJSONNoMemo|BenchmarkPigeonJSONOptimized|BenchmarkSt
dlibJSON)$' -run 'notests' .

goos: linux
goarch: amd64
pkg: github.com/mna/pigeon/examples/json
cpu: Intel(R) Core(TM) i7-10710U CPU @ 1.10GHz
BenchmarkPigeonJSONNoMemo-12                  64          17566232 ns/op
BenchmarkPigeonJSONOptimized-12              192           6306908 ns/op
BenchmarkStdlibJSON-12                      6556            217783 ns/op
PASS
ok      github.com/mna/pigeon/examples/json     5.402s
```

Wrap-up

pigeon & PEG - Lessons learned

- PEG is a helpful tool to on your tool belt for “esoteric” structured text (e.g. log and config formats).
- Separate Go code from PEG with helper functions.
- Write tests and iterate.
- Define an AST (abstract syntax tree).
- Start small and grow.
- Only add sophisticated error handling at the end.

Whois

Lucas Bremgartner

Go Engineer for hire (contractor)

Open source enthusiast
Active in the Go community since ~8 years
Co-Organizer of Bärner Go Meetup

Maintainer of (notable contributions):
rootcerts, logstash-config, errchkjson, bidichk, mna/pigeon, magnusbaeck/logstash-filter-verifier

Online:
github.com/breml | @_breml_ | linkedin.com/in/lucas-bremgartner/

Whois

Lucas Bremgartner

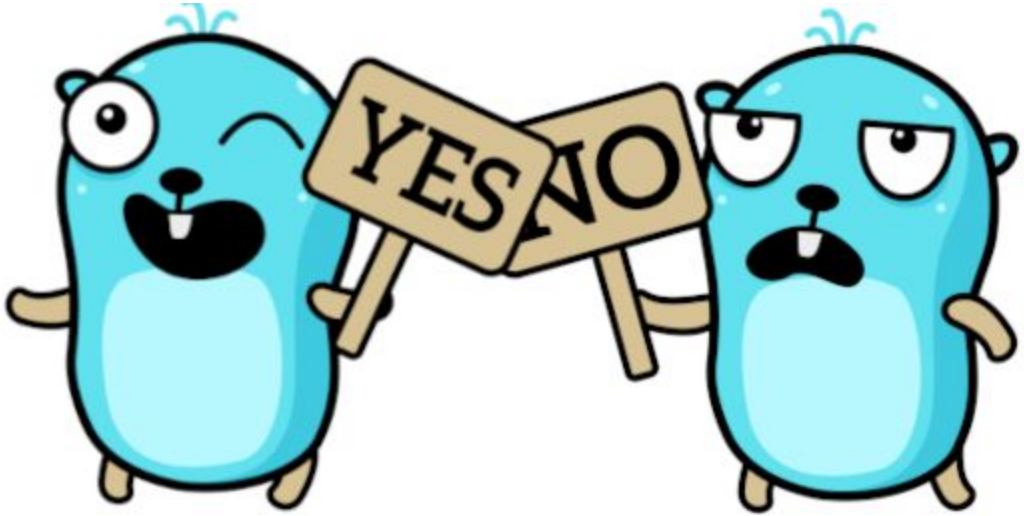
Go Engineer for hire (contractor)

Open source enthusiast
Active in the Go community since ~8 years
Co-Organizer of Bärner Go Meetup

Maintainer of (notable contributions):
rootcerts, logstash-config, errchkjson, bidichk, mna/pigeon, magnusbaeck/logstash-filter-verifier

Online:
github.com/breml | @_breml_ | linkedin.com/in/lucas-bremgartner/

Questions



Thank you

Links

PEG

Parsing Expression Grammars and Pappy by Bryan Ford
The Packrat Parsing and Parsing Expression Grammars Page | Mailing List

PEG.js Online Version

pigeon

github.com/mna/pigeon | Go reference