A Parsing Fancy

Parser Combinators in ~~Scala~~ Kotlin

Simon Gerber


WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL
TAP TAP

“A **domain-specific language** (DSL) is a computer **language** specialized to a **particular** application **domain**.”

Domain Specific Languages

- **Internal**: Legal syntax in the parent language. E.g. ~~Scala~~KotlinTest

“The tech-talk” should “teach me parser combinators”

Domain Specific Languages

- External: Roll your own
- Pro: Not constrained by another language’s grammar
- Con: Have to write your own parser

Parsing

Given a grammar and a string:

1. Is that string in the language of the grammar?
2. What is the *structure* of that string relative to the grammar?

Context-free Grammar

- Grammar ‘G’ describes a language ‘L’

- G = < T N S R >
  - T = Terminals (the words)
  - N = Non Terminals (phrases, expressions)
  - S = Start symbol (one of the non terminals)
  - R = Rules of the grammar

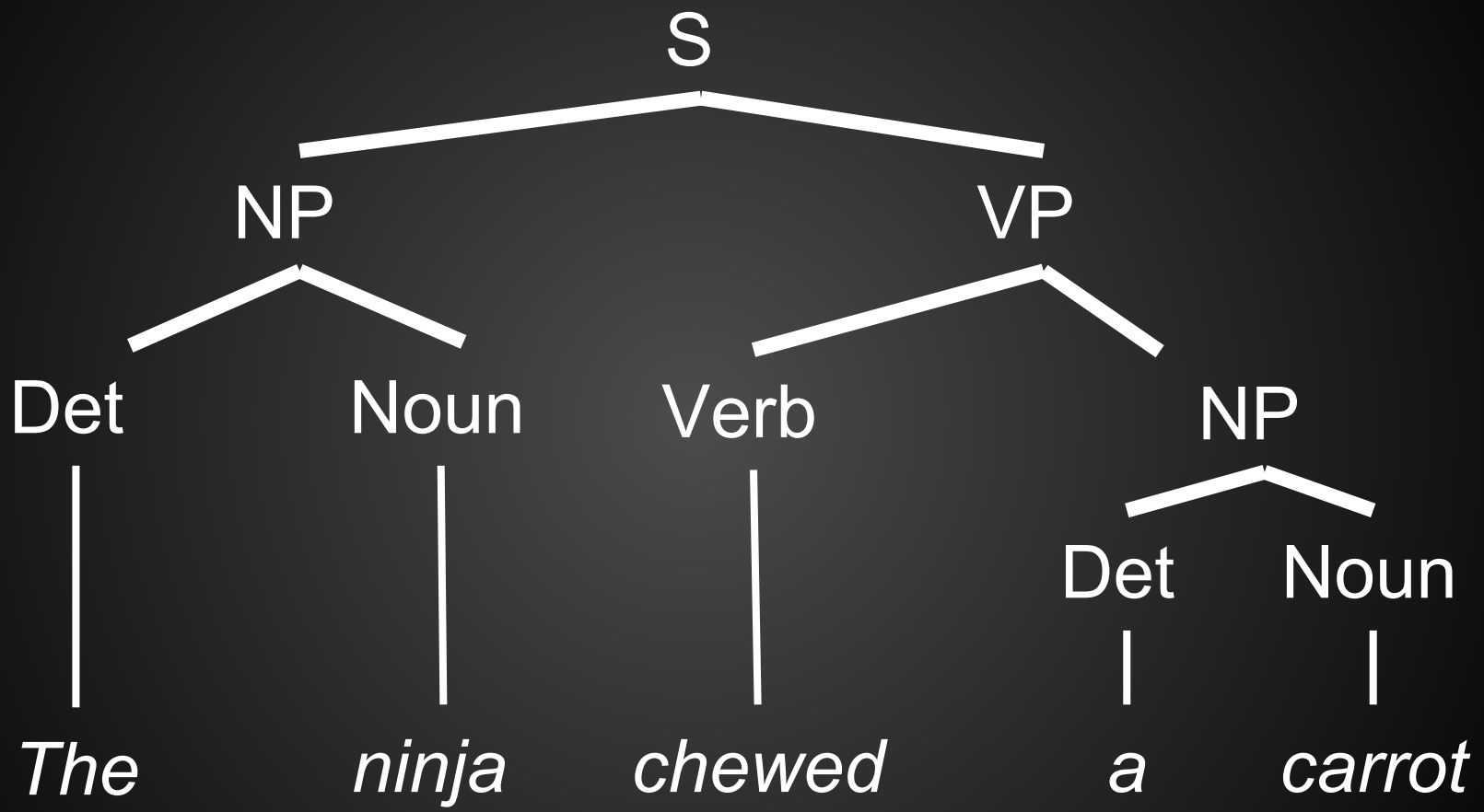
Context-free Grammar

- T = {a, carrot, chewed, coded, coder, googled, it, meal, ninja, the}
- N = {S, NP, VP, Det, Noun, Verb}
- S = S
- R = {

S → NP VP
NP→ Det Noun
VP → Verb
VP → Verb NP

Det → a | the | it
Noun → carrot | coder | ninja | meal
Verb → chewed | coded | googled

“The ninja chewed a carrot”


S
NP
VP
Det
Noun
Verb
NP
Det
Noun
The
ninja
chewed
a
carrot

Recursion in CFGs

PP → Prep NP

NP → Noun PP

[The programmer wrote his [$_{NP}$ program [$_{PP}$ with his pair

[$_{PP}$ with the Korean hair [$_{PP}$ from the team [$_{PP}$ that cares

[$_{PP}$ near the white kitchen [$_{PP}$ on the white floor

[$_{PP}$ in the noisy building [$_{PP}$ on Clarence street.]]]]]]]]]]

Parser Combinators

- Combine basic parsers to form complex rules
- ‘name → rule’ and composability from CFG

Parser Combinators in Kotlin

token(“\w+”) Parser<TokenMatch>

https://github.com/h0tk3y/better-parse

Parser Combinators in Kotlin

```
token(“\w+”)
Token asJust Cat()
```

```
Parser<TokenMatch>
Cat
```

https://github.com/h0tk3y/better-parse

Parser Combinators in Kotlin

```
token("\w+")
Token asJust Cat()
Token use { Cat(text) }
```

```
Parser<TokenMatch>
Cat
Cat(name = "Berlioz")
```

https://github.com/h0tk3y/better-parse

Parser Combinators in Kotlin

token(“\w+”)	Parser<TokenMatch>
Token asJust Cat()	Cat
Token use { Cat(text) }	Cat(name = “Berlioz”)
Cat and Dog	Tuple<Cat, Dog>

https://github.com/h0tk3y/better-parse

Parser Combinators in Kotlin

token(“\w+”)	Parser<TokenMatch>
Token asJust Cat()	Cat
Token use { Cat(text) }	Cat(name = “Berlioz”)
Cat and Dog	Tuple<Cat, Dog>
Cat and skip(Dog)	Cat

https://github.com/h0tk3y/better-parse

Parser Combinators in Kotlin

token(“\w+”)	Parser<TokenMatch>
Token asJust Cat()	Cat
Token use { Cat(text) }	Cat(name = “Berlioz”)
Cat and Dog	Tuple<Cat, Dog>
Cat and skip(Dog)	Cat
Cat or Dog	Carnivora

https://github.com/h0tk3y/better-parse

Parser Combinators in Kotlin

token(“\w+”)	Parser<TokenMatch>
Token asJust Cat()	Cat
Token use { Cat(text) }	Cat(name = “Berlioz”)
Cat and Dog	Tuple<Cat, Dog>
Cat and skip(Dog)	Cat
Cat or Dog	Carnivora
optional(Cat)	Cat?

https://github.com/h0tk3y/better-parse

Parser Combinators in Kotlin

token(“\w+”)	Parser<TokenMatch>
Token asJust Cat()	Cat
Token use { Cat(text) }	Cat(name = “Berlioz”)
Cat and Dog	Tuple<Cat, Dog>
Cat and skip(Dog)	Cat
Cat or Dog	Carnivora
optional(Cat)	Cat?
oneOrMore(Cat)	List<Cat>

https://github.com/h0tk3y/better-parse

Here’s one we prepared earlier...

https://github.com/sigerber/parsing-fancy_kotlin

https://github.com/sigerber/parsing-fancy (Scala)