Parsing For Dummies With Sly

—

Hadrien Titeux

Parsing: what computer scientists solved 40 years ago, but you still can't do it easily on your own*

**At least in Python*

The Problem

Typical parsing problems

- Some ad-hoc data description language
- Some small DSL language
- Anything recursive, with parentheses
- ...

Example 1: STL

```
solid cube_corner
  facet normal 0.0 -1.0 0.0
    outer loop
      vertex 0.0 0.0 0.0
      vertex 1.0 0.0 0.0
      vertex 0.0 0.0 1.0
    endloop
  endfacet
  facet normal 0.0 0.0 -1.0
    outer loop
      vertex 0.0 0.0 0.0
      vertex 0.0 1.0 0.0
      vertex 1.0 0.0 0.0
    endloop
  endfacet
```

```
  facet normal -1.0 0.0 0.0
    outer loop
      vertex 0.0 0.0 0.0
      vertex 0.0 0.0 1.0
      vertex 0.0 1.0 0.0
    endloop
  endfacet
  facet normal 0.577 0.577 0.577
    outer loop
      vertex 1.0 0.0 0.0
      vertex 0.0 1.0 0.0
      vertex 0.0 0.0 1.0
    endloop
  endfacet
endsolid
```

Example 2: SPARQLE

```
PREFIX ex: <http://example.com/exampleOntology#>
SELECT ?capital
       ?country
WHERE
  {
    ?x  ex:cityname       ?capital   ;
        ex:isCapitalOf    ?y         .
    ?y  ex:countryname    ?country   ;
        ex:isInContinent  ex:Africa  .
  }
```

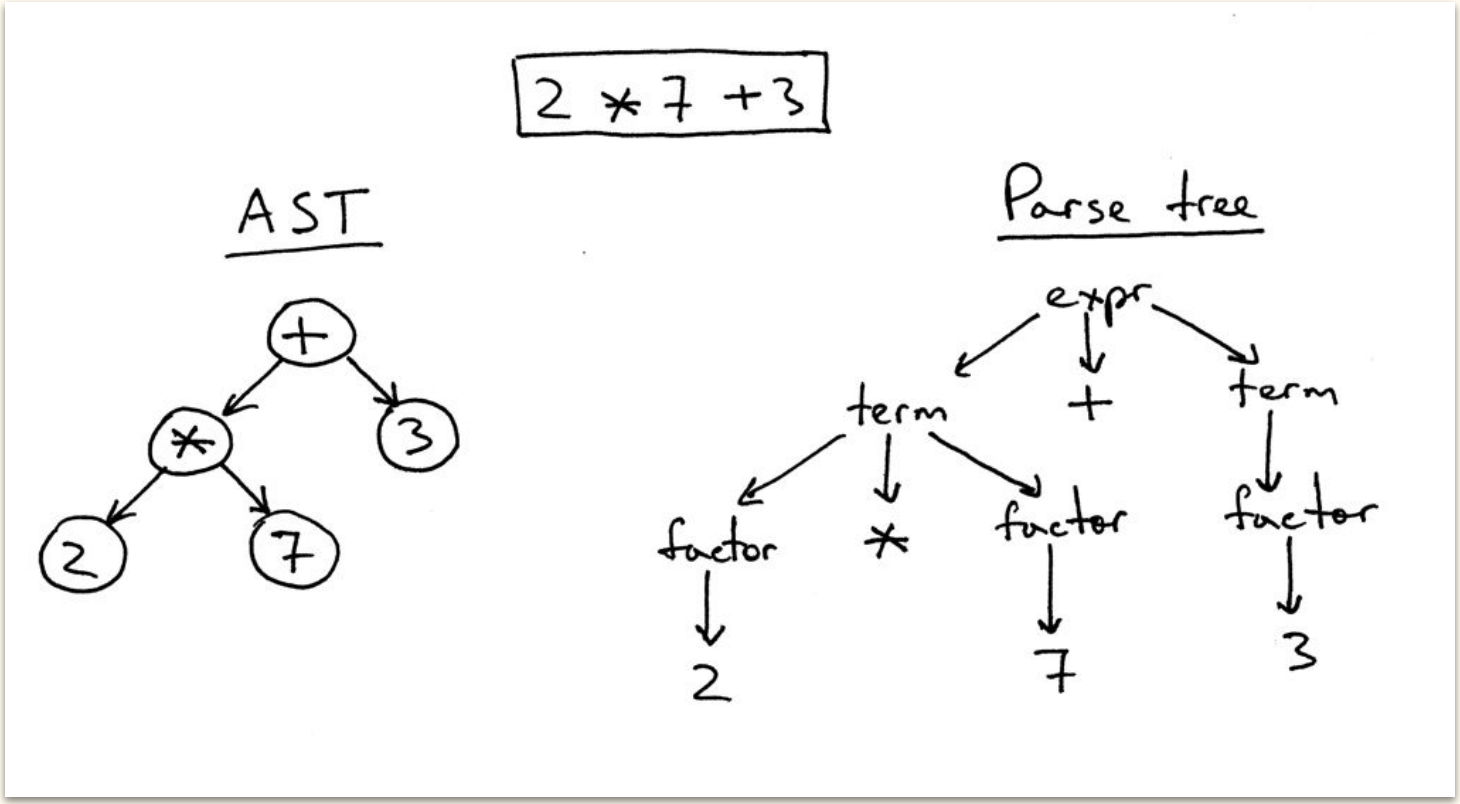
Example 3: CHAT annotations

```
@Begin
@Languages:     eng
@Participants: CHI Ross Child, FAT Brian Father
@ID:      eng|macwhinney|CHI|2;10.10||||Target_Child|||
@ID:      eng|macwhinney|FAT|35;2.||||Target_Child|||
*ROS:     why isn't Mommy coming?
%com:     Mother usually picks Ross up around 4 PM.
*FAT:     don't worry.
*FAT:     she'll be here soon.
*CHI:     good.
@End
```

Example 4: Simple Math

"((1 + 2 * 10) / 10) / 30 + 140 - 15.3"

Our goal


2 * 7 + 3
AST
+
*
3
2
7
Parse tree
expr
term
+
term
factor
*
factor
factor
2
7
3

The “parsing hell” mistake

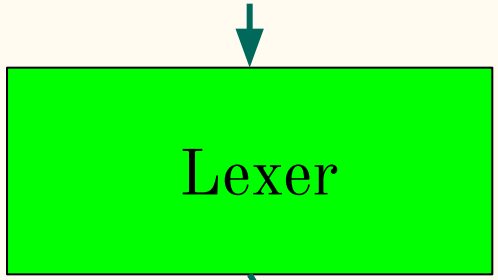
SLY

What is SLY

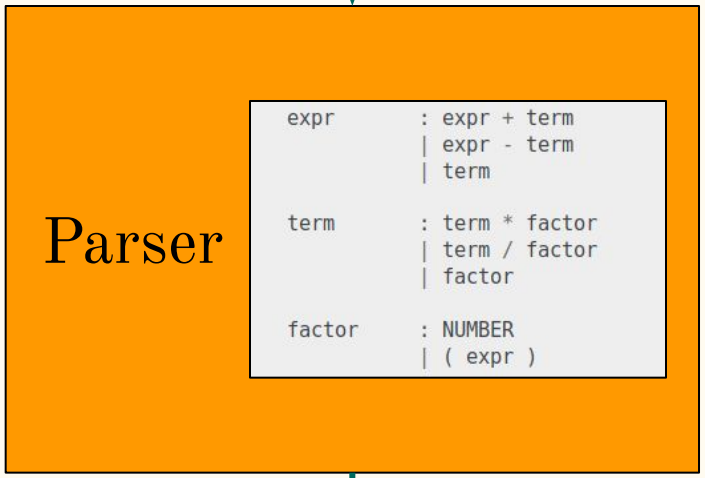
- A Python >=3.6 library
- A Python reimplementation of Yacc/Lecc
- A LALR(1) parser (c.f. Yacc)
- Easy to use
- Good at helping *you* figure out why your parser isn't working
- Good at helping you tell *the user* why their code is not valid
- Very helpful if you (like me) need an AST as an output

(Very) short primer on lexing/parsing

x = 3 + 42 * (s - t)




```
[ ('ID','x'), ('EQUALS','='), ('NUMBER','3'),
  ('PLUS','+'), ('NUMBER','42'), ('TIMES','*'),
  ('LPAREN','('), ('ID','s'), ('MINUS','-'),
  ('ID','t'), ('RPAREN',')') ]
```



AST

An example, you say?

Parsing JSON

Everyone knows JSON!

It has...

- Dictionary and lists, possibly nested
- Litterals (numbers or strings)

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

Step 1: figure out your grammar

- Try writing the BNR form of your code's grammar
- If you *reaaaaally* want to make sure it's going to work, check that it can be expressed in the Chomsky Normal Form (to ensure that it is *context-free*)

Step 1: figure out your grammar

```
json := object | array
object := '{' members '}'
members := pair | pair ',' members
array := '[' elements ']'
elements := value | value "," elements
pair := STRING ':' value
value := STRING | INTEGER | FLOAT | object | array
```

Step 2: Let’s look a the code

Finally: stuff we haven't seen

Ply can do tons of other very neat things:

- Proper error handling and syntax error recovery
- Parser rules debugging
- Dealing with ambiguous grammars using precedence rules
- In-parser AST construction
- Line number and position tracking
- And more subtle things...

Questions?