Pyrser Selector Language

Lionel Auroux

lionel@lse.epita.fr

2017-09-24
For pyconFr 2017 - Toulouse

Intro

About me

Researcher @ LSE (lse.epita.fr) in Security, System and Language

Teacher in CS @ EPITA and EPITECH:

- Python, Kernel for EPITA apprentissage
- Language Theory, Kernel for EPITECH (cycle master)
  Project KOOC (Kind Of Object C) :

  ```
  Made a compiler on POO Language
  Based an extended C Grammar
  One trimester to do the stuff
  Bac+3 Students
  ```

Pyrser?

Features of Pyrser

Pyrser is a toolbox for SLE (Software Language Engineering).

So with that, you could:

- Describe a grammar with a specific DSL
- Parse text with your grammar (PEG algorithm)
- Create an AST
- Type check it
- Transform it

Legacy

KOOC begin in 2003!

Requirements:

- Need something easier than LR parsers (grammar conflicts)
- Need a hackable C frontend (grammar composition)
- Workarounds

LL/LR

LR: *Left to right, Rightmost derivation* does a **Bottom-Up parse**.

LL: *Left to right, Leftmost derivation* does a **Top-Down parse**.

LR:

- we need to generate an **automata**.
- conflict could exists in grammar that disallow **automata** generation.
- complex to mix different grammar.

LL:

- just a bunch of recursive boolean function.
- **infinite loop** in left-recursive.

PEG (2004): correct LL default with a cache

Legacy

- 2003 perl Parse::Yapp (Lex/Yacc4perl)
- 2004-2012 Codeworker (LL(k))
- 2013-???? Pyrser (PEG):

Pyrser is all the good stuff of codeworker in python and more

Pyrser Selector Language?

The problem

Classical compilation scheme:

- Parsing -> Define a grammar -> AST
- Validating & AST Desugaring -> AST Visiting
- Type Checking -> AST Visiting
- Generation -> AST Visiting

Example

```
import ast
class Allnames(ast.NodeVisitor):
    def visit_Module(self, node):
        self.names = {}
        self.generic_visit(node)
        print(sorted(self.names.keys()))

    def visit_Name(self, node):
        self.names[node.id] = node

x = Allnames()
t = ast.parse('d[x] += v[y, x]')
x.visit(t)
```

The problem

By AST visiting:

- we visit only some type of node by pass
- we handle the states between different methods

Semantic = many passes, many states shared

So we need. . .

A centralized DSL (Domain Specific Language) to describe what to **match** and what to **transform**.

Like regexes, but for data structures.

PSL Basic

Design

What do we learn from regexes?

- Patterns
- Capture (Match group)
- Result

```
import re
...
g = re.match(
    "(\s+(?P<arg1>\w+)(\s*(?P<arg2>\S.*$)))?",
    inputs
    )
a1 = g.groupdict()["arg1"]
a2 = g.groupdict()["arg2"]
```

Design

What do we match?

- Type
- Value
- Attributes
- List (index)
- Dict (key)
- Strict or not (wildcards)

Design

What do we want to do?

- Reference on object to work on: **Capture**
- Call python code to do black magic: **Hook**

Hello, World

```
import pyrser.ast.psl as psl

def my_hook(capture, user_data):
    print("captured node %s" % repr(capture['a']))
    user_data.append(capture['a'])

class A: pass

parser = psl.PSL()
psl_comp = parser.compile("""
{
    A(...) -> a => #hook;
}
""")

user_data = []
t = [1, 2, A(), 3]
psl.match(t, psl_comp, {'hook': my_hook}, user_data)
```

Hello, World

- { ... } a block of many statement
- **...;** a statement
- **A(...)** match all **A** objects whatever there attributes
- -> **a** capture the object in a *register* named **a**
- => **#hook** call the hook **hook** bind to the **my_hook** function

Syntax tour

Type/Value:

- **A?()** match all **A** objects or subclasses, with no attributes
- **A(.a=42)** match all **A** objects with *only one* attribute **a** with value **42**
- **A(.a=4.2, . . .)** match all **A** objects with *at least one* attribute **a** with value **4.2**
- **A(.a=*, . . .)** match all **A** objects with *at least one* attribute **a** with *any* value
- **A(.*='42', . . .)** match all **A** objects with any attribute with value *'42'*

Syntax tour

List/Dict:

- **[2:A(. . .) -> a, . . .]** match all **A** at the **index 2** of an unknown object that implement ***.iter***
- **A({*:B(. . .) -> b, . . . }, . . .)** match all **B** at **any key** of an *object A* that implement **.keys()**

Syntax tour

Ancestor/Sibling:

- **A(. . .)** -> **parent** / **B(. . .)** / **C(. . .)** -> **leaf** match all **C** child of a **B** with a common parent **A**
- **A(. . .)** -> **lhs** ∼∼ **B(. . .)** -> **rhs** match all **A** and **B** that share a common parent
- **A(. . .)** / < **B(. . .)** ∼∼ **C(. . .)** > / **D(. . .)** guess it!

How it's works?

Ref

- Tree Automata Techniques & Application, October, 12th 2007 (aka Tata). Algorithm classification, proofs. . .

ok for algorithm proofs, but unusable as is. . .

- Pattern matching in tree structures, Thesis of Flouri Tomáŝ, September, 17th 2012, Czech Technical University

the solution. . .

In real life

XPath:

- **B/*[1]**: first child of B whatever its name

CSS Selector:

- **div > p**: all <p> child of <div>

But. . .

XPath, CSS Selector are Top-Down Tree Automata

Top-Down Tree Automata, so what?

Root are match befor leaf.

How to **transform** a tree if we can't change **leaf** BEFORE **roots**?

We need to walk our tree in reverse order -> **Bottom-Up order**.

The good Idea

In *Flouri tomáŝ* thesis, some advanced experimentation are done with an ad-hoc **LR algorithm** (**RR**?).

Is not really how **PSL** is implemented but it's very near.

Overview

Separation of concerns:

- Tree Walk in Bottom-Up order:
  - DFS: Deep-First Search
  - **yield from**
- Matcher automata
  - Validate sequence
  - Handle states, events, captures

Overview

Patterns looklike they are used in **Top-Down order**, it's a fake.

We generate **branches** in reverse order for **Bottom-Up matching**

branches are minimum unit of matching.

We extract **branches** that are match in parallel during **tree traversal**.

We **validate** sequence of collective **branches** that are part of the same **pattern**.

Conclusion

Release

Dev branch 0.3.0 (with PSL)

```
$ git clone https://github.com/LionelAuroux/pyrser
```

Stable branch 0.2.0 (without PSL)

```
$ pip install pyrser
```

Documentation

http://pythonhosted.org/pyrser

Help

- Feedbacks
- Documentation! (my english is broken)
- Documentation of PSL (Work in progress, Read the Test)

Questions

Q/A!