DSL with pyrser

Author: L. Auroux

lionel@lse.epita.fr
For pyParis 2018

Quick summary

- About Domain Specific Modeling/Language
- About Compiler creation...
- ...in python
- About Pyrser

About Domain Specific Modeling/Language

Each domain have his own words, relying on his own concepts.

```
If I'm selling to you, I speak your language.
If I'm buying, Alors vous devez me parler en Français
```

thanks Willy Brandt

About Domain Specific Modeling/Language

DSM literally follow this principe by promoting the design of DSL to mimic words and concepts of domain.

- Domain: system, class of problems
- DSM: Domain Specific Modeling
- DSL: Domain Specific Language

About Domain Specific Modeling/Language

So:

- No more specification in human language
- Get a formal language for the Domain

And:

- Words become Abstractions
- Concepts become Algorithm
- DSL as direct input for ad-hoc tools

About Compiler creation...

Two way to create DSL.

1. Embedded DSL (use a host language)

```
from scapy.all import *
# ...
ether = Ether(dst="ff:ff:ff:ff:ff:ff")
ip = IP(src="0.0.0.0",dst="255.255.255.255")
udp = UDP(sport=68,dport=67)
bootp = BOOTP(chaddr=hw)
dhcp = DHCP(options=[("message-type","discover"),
    "end"])
dhcp_discover = ether / ip / udp / bootp / dhcp

ans, unans = srp(dhcp_discover, multi=True,
    timeout=5)
```

About Compiler creation...

2. True Compiler/Interpreter

Anatomy of a compiler

- Grammar -> Parsing -> AST
- Handle AST:
  - semantic
  - typing
- Interpretation / Code generation

Type of grammar

- CFG (Context Free Grammar)
  - Production rules -> Automata
  - Token (scanner)
  - Parser
- PEG (Parsing Expression Grammar) (2004)
  - Scannerless
  - Top-down recursive parser with memoization
    - so Rules are functions/methods
  - Priority choice

. . . in python

- CFG (Context Free Grammar)
  - PLY
  - PlyPlus
  - Lrparsing
  - . . .
- PEG (Parsing Expression Grammar)
  - Arpeggio (Aug 2014)
  - Parsimonious (Dec 2012)
  - Tatsu (May 2017), Grako (Jun 2013)
  - Pyrser (Aug 2013)
  - . . .

About Pyrser

A bit of history

Epitech KOOC Project (2013-2017): Kind Of Object C.

- Student must create a superset of C language with classes (CFront revival).
- Compiler write in pyrser (Cnorm)
- Compiler product C

Why another tool?

What other tools do that bother me:

- Automatic CST (parse tree) creation
- Provide only features for parsing
- Mix grammar and host language (action)
- Python3
- CFG! 2013

Pyrser main features

```
iopi$ pip3 install pyrser
```

- Parsing:
  - Basic classes provide PEG Parser in a EDSL way
  - BNF like language to write Grammar
- Tree handling:
  - PSL (Pyrser Selector Language)
  - Tree matching and rewriting
- Type checking:
  - You have module for type check your language.

Grammar examples

1 CSV parser

```
from pyrser import grammar
class Csv(grammar.Grammar):
    entry = "csv"
    grammar = """
        csv = [
            @ignore("null") [ line eol ]+
            line? eof
        ]

        line = [
            item [';' item ]*
        ]

        item = [ [ ~[';' | eol] ]* ]
    """
```

Grammar examples

- Grammar is a Class
  - so inheritable (grammar composition)
- Rule are Method
  - so overidable

```
class A(grammar.Grammar):
    grammar="""
        rule = [ id eof ]
    """

class B(grammar.Grammar, A):
    grammar="""
        rule = [ [ A.rule | string ] eof ]
    """
```

Grammar examples

- 2 abstractions two handle AST:
  - Nodes for data handling
  - Hooks for event handling

```
// inside a DummyGrammar
R = [
    ThisRuleReturnSomethingIn_ : weCaptureInThisNode
]

ThisRuleReturnSomethingIn_ = [
    #putSomethingIn(_)
]
```

- weCaptureInThisNode is a **Node**
- _ is the returning **Node** of the **current** Rule
- #putSomethingIn is **hook**

Grammar examples

Defining hooks outside the class **DummyGrammar** definition.

```
from pyrser import meta

@meta.hook(DummyGrammar)
def putSomethingIn(self, _):
    _.is_touched = True
    return True
```

Grammar definition

```
!expr:
    Negative lookahead.
    Fails if the next item in the input matches expr.
    Consumes no input.
!!expr:
    Positive lookahead.
    Fails if the next item in the input does not matches expr.
    Consumes no input.
~expr:
    Complement of expr.
    Consumes one character if the next item in the input matches does not matches expr.
->expr:
    Read until expr. Consumes N character until the next item in the input matches expr.
A:
    Call the rule A.
'a':
    Read the character a in the input.
"foo":
    Read the text foo in the input.
'a'..'z':
    Read the next character if its value is between a and z.
...
```

Grammar examples

More complete examples:

- How to create a JSON parser:

https://pythonhosted.org/pyrser/tutorial1.html

- A complete C Frontend:

https://github.com/LionelAuroux/cnorm
https://pythonhosted.org/cnorm/

Pyrser Selector Language

PSL describe what to **match** and what to **transform**

```
import pyrser.ast.psl as psl

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

Pyrser Selector Language

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

class A: ...

user_data = []
t = [1, 2, C(v=A()), {'toto': A(flags=True)}]
psl.match(t, psl_comp, {'hook': my_hook}, user_data)
```

Pyrser Selector Language

What do we match?

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

Pyrser Type System

Pyrser provides a basic "type system" module to check any producted AST.

Due to KOOC project, this TS focus on ad-hoc polymorphism.

No type reconstruction yet.

```
from pyrser.type_system import *

t1 = Type('int')
t2 = Type('double')
var = Var('var1', 'int')
f1 = Fun('fun1', 'int', [])
f2 = Fun('fun2', 'int', ['char'])
f3 = Fun('fun2', 'int', ['int', 'double'])
scope = Scope(sig=[t1, t2, var, f1, f2, f3])
print(str(scope))
```

Pyrser Type System

```
scope :
    type double
    fun fun1 : () -> int
    fun fun2 : (char) -> int
    fun fun2 : (int, double) -> int
    type int
    var var1 : int
```

Pyrser provide technics to connect AST to inference:

https://pythonhosted.org/pyrser/tutorial3.html

Roadmap

- KOOC will evolve in KOOC++
- So, Pyrser needs too
  - An agnostic version of PSL: treematching (WIP)
  - A better TS (wand's Type Inference Algo)

Conclusion

Q/A!

- **slides**
- `https://github.com/LionelAuroux/pyrser`