Metamodel-based Parser Generator for Python (PyCo)

Author: Ing. Sharoon Ilyas
Supervisor: Ing. Sergej Chodarev PhD.

Technical University of Košice
Faculty of Electrical Engineering and Informatics

Motivation For Creating PyCo

1. A tool that uses object-oriented approach to define a model centred Domain Specific Language (DSL) in Python.
2. Abstract syntax of the DSL can be expressed using object-oriented model and concrete syntax for the DSL can be expressed using decorators and field function of data-classes Python library.
3. Meta-model analysis.
4. EBNF (extended BNF) grammar generation.
5. Class instances generation using PyCo.

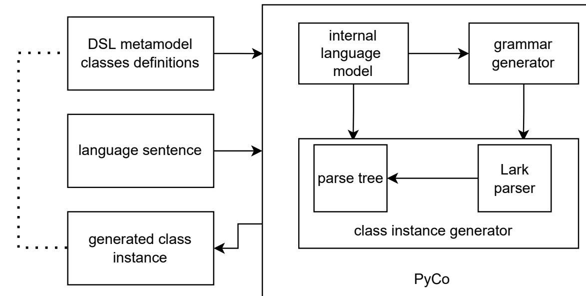
Method And Implementation

1. Classes in Python are defined representing the language abstract syntax.
2. Concrete syntax is defined using fields function of dataclasses library.
3. PyCo takes the specified root class and finds the relationships and inheritance along with the field types and properties recursively and generate Extended Backus–Naur form (EBNF) grammar and store metadata related to the domain specific language model.
4. The grammar is taken by the Lark parser resulting in parse tree.
5. Generated parse tree is parsed and metadata which was generated in the previous steps are used to create the class instances recursively.

Requirements for PyCo

1. Minimum required Python version is 3.7
2. Required version of Lark Python library is 1.1.1

PyCo Component Structure



Robot Domain Specific Language Model

```
from abc import ABC
from dataclasses import dataclass, field
from typing import List
from pyco.utils.decorators import syntax

class Commands(ABC):
   pass

@dataclass
@syntax(before="turn")
class Turn(Commands):
   direction: str = field(metadata={'token': 'DIRECTION_TOKEN'})
   speed: float = field(metadata={'token': 'SPEED_TOKEN'})

@dataclass
@syntax(before="begin", after='end')
class Robot:
   body: List[Commands] = field(metadata={'separator': ','})
```

Robot Domain Specific Language Implementation

```
from examples.robot.classes import Robot
from pyco.pyco_meta_parser import PyCo, NUMBER_REG

token = {
   "DIRECTION_TOKEN": '"back" -> back|"right" -> right|"left" ->left',
   "SPEED_TOKEN": NUMBER_REG,
}

if __name__ == '__main__':
   pyco = PyCo(Robot, token)
   instance = pyco.parse('begin turn left 40 , turn right 50 end')
   print(instance)
```

Result: Robot(body=[Turn(direction='left', speed=40.0), Turn(direction='right', speed=50.0)])

Conclusion

The tool is capable of creating various domain specific language and can handle unlimited depth of class relationships in the metamodel of the domain specific language. The tool not only analyse the metamodel and generate all the metadata required for its needs, but also infer the needed metadata.

The tool also gives programmer the ability to see the parse tree and check the generated grammar helping him with in depth insight of the structure the programmer is trying to build.

With the usage of this tool, the generation of DSL has become very fast and simple and the programmer saves a lot of time and effort.

Limitations of PyCo

1. To avoid ambiguity in grammar, token names should be different than the defined class names in Domain Specific language model.
2. PyCo does not support infix operators with priority levels, thus domain specific languages related to mathematics are not possible in the current version of PyCo.
3. PyCo relies on the error handling of Lark library only.

Future Implementations of PyCo

1. Infix operators can be included in PyCo to implement mathematical based domain specific languages.
2. Lark parser has many filters and search methods to parse the parse tree, which can be used to increase and implement various functionalities in PyCo.
3. Strong error handling can be added to PyCo, making it easier to programmers to debug and write code more efficiently.
4. Other parsing algorithms by Lark can be explored to handle ambiguous grammars too.