神奈川大学理学部　情報科学科　馬谷研究室

ブロックを記述可能なラムダ式の CPython 上の実装

青木 宇宙 201803357

1　はじめに

CPython は C 言語を使って書かれた Python 処理系である．最もよく使われている Python 実装であり，オリジナルの Python 処理系でもある．現在の Python にはラムダ式の本体にブロックを書くことが出来ない，という問題がある．そこで本研究では抽象構文木の変換を用いてラムダ式の本体にブロックを書くこと可能にすることを目的とする．ブロックを書くことができるようになると，複雑な処理をラムダ式本体中に書くことが可能になる．

2　背景

まず，Python のラムダ式について簡単に説明する．ラムダ式とは無名関数と呼ばれる関数を書くための記法で，関数に名前を付けずに簡潔に宣言できるというメリットがある．ただし，他のプログラミング言語と異なり，現在の Python ではラムダ式の本体にブロックを書くことが出来ない．本研究では，CPython を用いてラムダ式の本体にブロックを書けるようにする．

次に，Python のラムダ式の文法について説明する．現在の Python では以下のように 1 行で書くことができる．`x + y` の部分が返り値を指定する式であるため，ここにブロック式を書くことができない．

```
>>> lam = lambda x,y: x + y
>>> print(lam(1,2))
3
```

それに対し，本研究では次のように書けるようにする．

```
>>> lam = lambda x:
            y = x + 1
            return x * y
>>> print(lam(2))
6
```

さらに，条件分岐 (if 文) や，繰り返し処理 (for 文) のような，より複雑な処理をラムダ式本体中に書くことも可能である．

3　CPython

本研究では，前節で説明した拡張ラムダ式を CPython 処理系に実際に実装する．CPython 処理系内部の処理の全体構成，ラムダ式と関数定義文の抽象構文木についての説明を行う．

初めに，CPython はソースコードから抽象構文木（AST）を生成するまでのフロントエンドと，AST をバイトコードへ変換・実行するバックエンドの大きく 2 つに分かれている．AST とは，プログラムの文法構造を木で表したものである．本研究では，拡張ラムダ式を含んだ AST を，拡張ラムダ式を含まない AST に変換してから，従来の方法によりに実行することで，拡張ラムダ式を実現する．

次にラムダ式と関数定義文の AST について説明する．関数定義文の AST は例えば，

```
def f(x,y):
    return x+y
```

のような関数定義は，以下の AST に変換される．

```
FunctionDef(
    name='f',
    args=arguments(
        posonlyargs=[],
        args=[
            arg(arg='x'),
            arg(arg='y')],
                /* 省略 */
    body=[
```

```
        Return(
            value=BinOp(
                /* 省略 */
```

それぞれ， name は関数名， args は引数， body は関数の中身である．それに対して，ラムダ式の AST は以下の通りである．

```
Expr(
    value=Lambda(
        args=arguments(
            posonlyargs=[],
            args=[
                arg(arg='x'),
                arg(arg='y')],
                /* 省略 */
        body=BinOp(
            /* 省略 */
```

拡張ラムダ式の AST は既存のラムダ式の AST とは body の型が異なっており，関数定義の body と同じにすることでブロックを書けるようにしている．

4 実装方法

本研究では，ブロックを記述可能なラムダ式を実装するために下記の手順に従って CPython を修正する．

1. Grammar/python.gram
   文法の定義が書かれたファイルで，PEG(Parsing Expression Grammar) 記法を採用している．ここでは，ラムダ式の書き方などの定義を行った．
2. Parser/Python.asdl
   構文解析部分を自動生成するために使用されるファイルで，python.gram 同様 PEG 仕様を採用している．ここでは body の型を stmt 型に変更し，返り値を格納する returns，型をコメントとして記述したオプションの文字列である type_comment を追加する．
3. Python/ast.c
   AST の生成に使用されるファイルである．ここでは，python.asdl で変更や追加した引数に構文木を生成するようなコードを追加する．
4. Python/pythonrun.c
   入力から構文解析器とコンパイラを実行するのに使用されるファイルである．本研究では，AST の変換を用いてブロックを記述可能なラムダ式を実装するため，まず AST を循環するための関数を作成し，その引数に巡回している階層と行数を格納するものを定義する．巡回している行にラムダ式があった場合，同じ階層の 1 つ上の行に関数定義を追加し，さらにそれを実行するような AST を返す．具体的には，

```
a = lambda(x):
    y = x + 1
    return x * y
```

のコードは，

```
def f(x):
    y = x + 1
    return x * y
a = f
```

へと変換される．

5 まとめ

本研究では，ブロックを記述可能なラムダ式の CPython 上の実装を行った．ラムダ式の AST の body を関数定義文の AST の body と同じにすることでブロックを記述可能にした．実装することでラムダ式の本体にブロックを書くことが出来ないという問題を解決した．

参考文献

[1] Python Software Foundation, Python 3.9.4 (https://docs.python.org/ja/3.9/)
[2] Anthony Show, CPython Internals: Your Guide to the Python 3 Interpreter, Real Python, 2021.