SimpliPy: A notional machine for learning Python

International Insitute of Information Technology Hyderabad

Venkatesh Choppella, Moida Praneeth Jain, Gnaneswar Kulindala, Prabhav Shetty, Anushka Srikanth

Notional Machine for Python with Procedures

Syntax

- Three new instructions:
  - Function definition
  - Return
  - assignment/call

- All function bodies end with a `return` instruction.

```python
def f(x):
    x = 5
    y = 10
    return x + y
a = f(2)

```

Note that expressions are simple, i.e., they do not include function calls.

What is the structure of this program?

```python
x = 5
y = 10
def f(z):
    if z > 5:
        x = 10
    else:
        x = 20
    return x + y + z
a = f(2)
```

Lexical Blocks

```python
x = 5
y = 10
def f(z):
    if z > 5:
        x = 10
    else:
        x = 20
    return x + y + z
a = f(2)

```

- A function body is a special block, called a **lexical block**.
- The top level block is also a **lexical block**.

Free vs Bound Variables

```python
x = 5
y = 10
def f(z):
    if z > 5:
        x = 10
    else:
        x = 20
    return x + y + z
a = f(2)

```

- A variable is **free** with respect to a lexical block if it is not *declared* in the block but occurs in any of its expressions.

- A variable is **declared** with respect to a lexical block if it is *declared* in the block.

decvars of a lexical block

```python
x = 5
y = 10
def f(z):
    if z > 5:
        x = 10
    else:
        x = 20
    return x + y + z
a = f(2)

```

- The decvars of a lexical block are the set of identifiers *declared* in that lexical block.

Control Transfer Functions: call and ret

```
0  def f(x):
1      x = 5
2      y = 10
3      return x + y
4  a = f(2)
5
```

Loc	next	call_0	ret_4	err
0	4	-	-	-
1	2	-	-	5
2	3	-	-	5
3	-	-	5	5
4	5	1	-	5
5	-	-	-	-

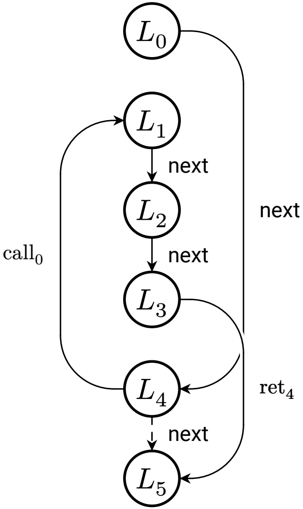
Why do we need multiple `calls` and `rets`?

```
def f():
    return 2
def g():
    return 4
if False:
    a = f
else:
    a = g
z = a()

```

Procedural: Control Flow Graph

```
0  def f(x):
1      x = 5
2      y = 10
3      return x + y
4  a = f(2)
5
```



Do we need more than one environment?

```
x = 5
y = 10
def f(z):
    x = 2
    return x + y + z
x = x + 1
a = f(2)

```

Lexical Map

$$\text{LexicalMap} : \text{EnvId} \rightarrow \text{Env}$$

Consider a lexical map e, with two environments

$$e_0 = \{x \mapsto 1, y \mapsto 5\}$$

$$e_1 = \{z \mapsto 10, y \mapsto 5\}$$

$$e = \begin{cases} 0{:}\{x{\mapsto}1,y{\mapsto}5\} \\ 1{:}\{z{\mapsto}10,y{\mapsto}5\} \end{cases}$$

How do we keep track of these environments?

```
x = 5
y = 10
def f(z):
    x = 2
    return x + y + z
x = x + 1
a = f(2)

```

Parent Chain

$$\mathrm{ParentChain} : \mathrm{EnvId} \rightharpoonup \mathrm{EnvId}$$

Consider the parent chain

$p[1] = 0$
$p[2] = 0$
$p[3] = 2$

How do we lookup/update a variable?

$$\text{lookup}^* : \text{Id} \times \text{LexicalMap} \times \text{ParentChain} \times \text{EnvId} \to \text{Res}$$

Consider the lexical map

$e_0 = \{x \mapsto 5, y \mapsto 10\}$

$e_1 = \{x \mapsto 20\}$

$e_2 = \{y \mapsto 20\}$

$e_3 = \{x \mapsto 11, z \mapsto 5\}$

and the parent chain

$$p = [(1, 0), (2, 0), (3, 2)]$$

How do we construct the parent chain?

```python
x = 5
y = 10
def f(z):
    x = 2
    return x + y + z
x = x + 1
a = f(2)

```

Closures

$$\text{Val} = \text{Num} + \text{Bool} + \text{Str} + \text{Closure}$$

$$\text{Closure} = \text{Loc} \times \text{EnvId} \times \text{Formals}$$

$$\text{Formals} = \text{List}[\text{Id}]$$

```
0  x = 5
1  y = 10
2  def f(z):
3      x = 2
4      return x + y + z
5  x = x + 1
6  a = f(2)
7
```

How do we know where to return to?

```python
x = 5
y = 10
def f(z):
    x = 2
    return x + y + z
x = x + 1
a = f(2)

```

Context

$$\text{Context} = \text{Loc} \times \text{EnvId}$$

```python
x = 5
y = 10
def f(z):
    x = 2
    return x + y + z
x = x + 1
a = f(2)

```

Continuation

$$\text{Continuation} = \text{List}[\text{Context}]$$

```python
x = 5
y = 10
def f(z):
    x = 2
    return x + y + z
x = x + 1
a = f(2)

```

State of the Machine

$$\text{State} = \text{LexicalMap} \times \text{ParentChain} \times \text{Continuation}$$

$$(e, p, k)$$

Tranisitions of the Machine

$$(e, p, k) \xrightarrow{\text{tick}} (e', p', k')$$

Function Definition Transition

- Construct closure

- Update the lexical map

```
def f(x):
    y = 2
    return x + y
a = f(2+3)
```

Function Call Transition: Step 1

Lookup the function

```python
def f(x):
    y = 2
    return x + y
a = f(2+3)

```

Function Call Transition: Step 2

Evaluate the arguments

```
def f(x):
    y = 2
    return x + y
a = f(2+3)

```

Function Call Transition: Step 3

Create environment from the decvars

```python
def f(x):
    y = 2
    return x + y
a = f(2+3)

```

Function Call Transition: Step 4

Update the lexical chain

```python
def f(x):
    y = 2
    return x + y
a = f(2+3)

```

Function Call Transition: Step 5

Push new context to continuation

```
def f(x):
    y = 2
    return x + y
a = f(2+3)
```

Return Transition

- Evaluate expression
- Pop the continuation
- Assign the result

```python
def f(x):
    y = 2
    return x + y
a = f(2+3)

```

Run of the Machine

```
def f(x):
    x = 5
    y = 10
    return x + y
a = f(2)

```

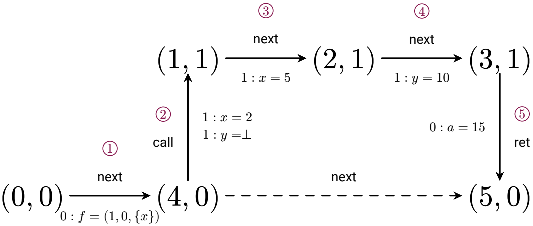
Execution Diagram

```python
def f(x):
    x = 5
    y = 10
    return x + y
a = f(2)

```

Execution Diagram

```
0  def f(x):
1      x = 5
2      y = 10
3      return x + y
4  a = f(2)
5
```



$$e_0 = \begin{cases} f \mapsto (1, 0, \{x\}) & \textcircled{1} \\ a \mapsto 15 & \textcircled{5} \end{cases}$$

$$\begin{tikzcd} 0 \\ 1 \arrow[u] \end{tikzcd}$$

$$e_1 = \begin{cases} \cancel{x \mapsto 2} & \textcircled{2} \\ \cancel{y \mapsto \perp} & \textcircled{2} \\ x \mapsto 5 & \textcircled{3} \\ y \mapsto 10 & \textcircled{4} \end{cases}$$

Summary

- Function definition, return and call instructions
- `call` and `ret` control transfer functions
- Lexical Map
- Parent Chain
- Closures
- Context and Continuation
- Execution Diagarm