CIS 624 Programming Languages

Dave Patterson, Walt O'Connor, Matt Hall

Parser

Sly parser generator to parse the input code

Returns an abstract syntax tree (AST)

EBNF

- EBNF describes construction of the parser
- Initial EBNF had conflicts that were not translatable to our implementation

```
program     ::=  { statements }
statements  ::=  statement { stmtsep statements }
statement   ::=  expression | assignment | if_stmt | for_stmt | zerosfunc
if_stmt     ::=  if conditional stmtsep statements
                 { stmtsep elseif conditional stmtsep statements }
                 [ stmtsep else statements ]
                 stmtsep end
for_stmt    ::=  for assignment stmtsep statements stmtsep end
zerosfunc   ::=  zeros | zeros() | zeros(digits) | zeros(matrixidx)

assignment  ::=  variable = expression
expression  ::=  number | string | operation | conditional | range |
                 vector [ ( vectorindex ) ] | matrix [ ( matrixindex ) ]
variable    ::=  word | vector [ ( vectorindex ) ] | matrix [ ( matrixindex ) ]
```

Operational Semantics

- Modeled semantics scope with “stack of heaps (frames)”
- Heaps are implemented as Python dictionaries
- Stacks are Python lists

Search Stack 1

$$\frac{name \in frames}{Stack : name \rightarrow Stack : value}$$

Search Stack 2

$$\frac{name \notin frames}{Stack : name \rightarrow Stack : skip}$$

Update Stack

$$\frac{}{Stack : name , value \rightarrow Stack ; name \Rightarrow value ; skip}$$

push_frame

$$\frac{}{Stack , frame \rightarrow Stack + frame ; skip}$$

pop_frame

$$\frac{Statck \rightarrow head , tail}{Stack \rightarrow head ; skip}$$

Interpreter

- Evaluate left and right expression for side effects
- Compute the value of the binary expression using the cached values from the left and right expressions
- Store that result and return the stack with the side effects from the left and right trees applied

Binary Expression Evaluation

```
def __init__(self, left: Expr, op, right: Expr):
      self.left = left\ self.right = right\ self.op = op

def eval(self, ctx):
      ctx2 = self.left.eval(ctx)
      ctx3 = self.right.eval(ctx2)
      …
      self.result = ops[self.op](self.left.get_value(),
      self.right.get_value())
      return ctx3

def get_value(self):
      return self.result
```