Interoperability between Python and Scheme Syntax using AST Manipulation

Principles of Programming Languages

Presented By : Mohammad Zaid

10 December, 2024

Motivation and Overview

- Motivation : Need seamless integration of Scheme-like syntax and Python code.

- Overview : Dynamically convert Scheme-like syntax into Python ASTs for runtime execution.

Objectives

- Dynamic Conversion : Scheme-like → Python AST

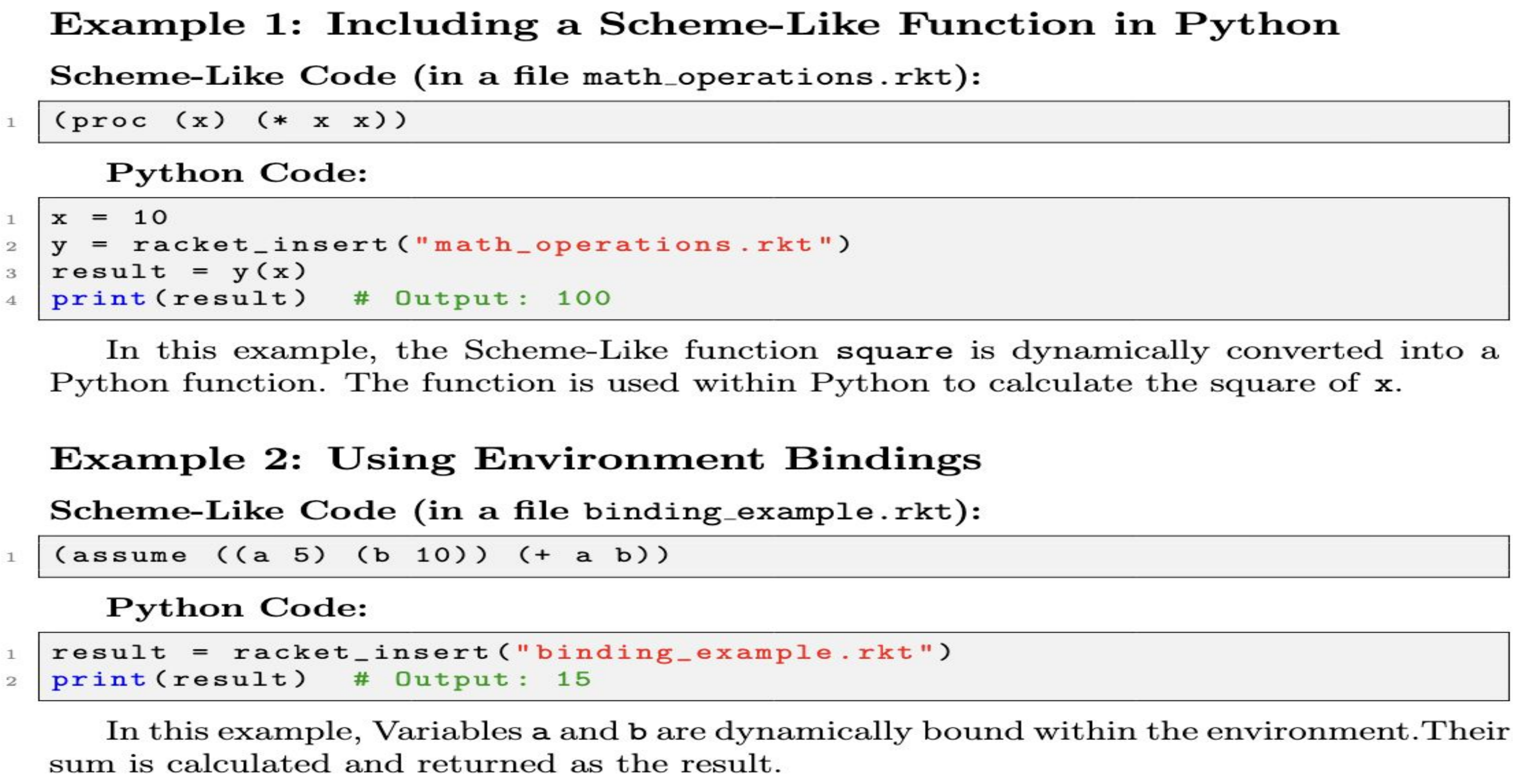
- Interoperability : Use Scheme-like modules/functions in Python as if they're native

- Flexibility : Combine functional constructs with Python's procedural/OO features

Supported Scheme-like Syntax

- Program ::= Expression
- Expression ::= Number | Identifier | (operator Expression Expression) | (assume ((Identifier Expression)...) Expression) | (proc Identifier Identifier Expression) | (if Expression Expression Expression)
- Numbers ::= A Numeric Literal
- Operators ::= + , - , * , / , < , > , =
- Bindings : (assume ((Identifier Expression) ...) Expression)
- (proc Identifier Identifier Expression)
- (if Expression Expression Expression)

Examples



Example 1: Including a Scheme-Like Function in Python

Scheme-Like Code (in a file math_operations.rkt):

```
(proc (x) (* x x))
```

Python Code:

```
x = 10
y = racket_insert("math_operations.rkt")
result = y(x)
print(result)  # Output: 100
```

In this example, the Scheme-Like function **square** is dynamically converted into a Python function. The function is used within Python to calculate the square of **x**.

Example 2: Using Environment Bindings

Scheme-Like Code (in a file binding_example.rkt):

```
(assume ((a 5) (b 10)) (+ a b))
```

Python Code:

```
result = racket_insert("binding_example.rkt")
print(result)  # Output: 15
```

In this example, Variables a and b are dynamically bound within the environment.Their sum is calculated and returned as the result.

Example 4: Environment with Multiple Bindings and Arithmetic

Scheme-Like Code (in a file `arithmetic_env.rkt`):

```
(assume ((x 3) (y 16)) (+ x y))
```

Python Code:

```python
x = 10
y = 15
result = racket_insert("arithmetic_env.rkt")
z = x * result
print(z)  # Output: 190
```

In this example,the `assume` construct binds `x = 3` and `z = 4`. These bindings are local to the Scheme-like code. The variable `y` is not defined in the Scheme-like code, so it refers to Python's globally defined `y = 15`. The Scheme-like code computes `(+ x y)` using its local `x = 3` and Python's global `y = 15`, resulting in `3 + 15 = 18`. This result is returned to Python as `result`.

Example 5

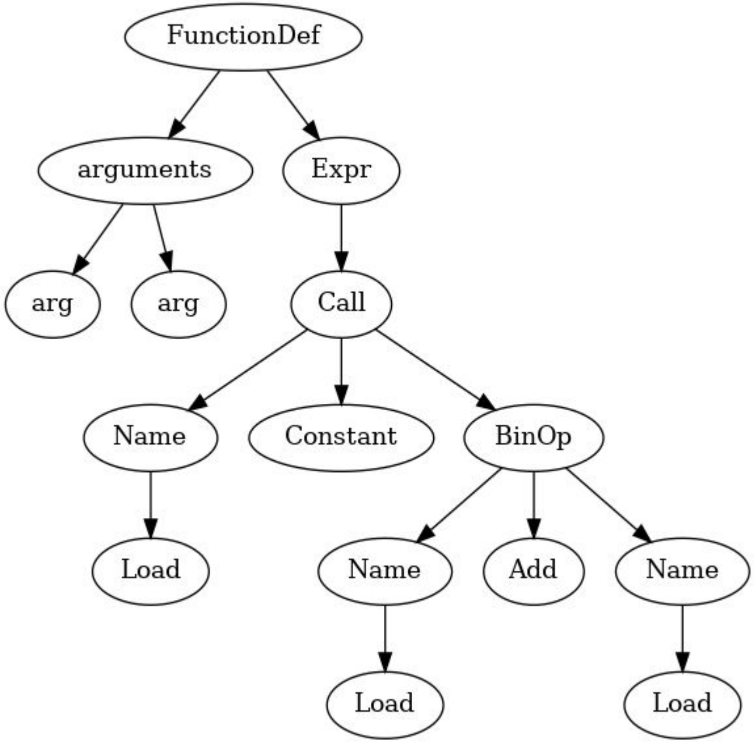
Scheme-Like Code (in a file `polynomial_example.rkt`):

```
(proc (a b c x) (+ (* a (* x x)) ( + (* b x) c)))
```

Python Code:

```python

funcc = racket_insert("polynomial_example.rkt")

# Manipulate the polynomial mathematically
scale_factor = 2  # Scale the polynomial's coefficients
shift_value = 3  # Shift the polynomial vertically

# Redefine polynomial with scaled coefficients
def transformed_quadratic(a, b, c, x):
    global funcc
    global shift_value
    return scale_factor * funcc(a, b, c, x) + shift_value

x_val = 4
result = transformed_quadratic(1, 2, 3, x_val)
print(result)  # Output : 57
```

Dynamic Linking Workflow



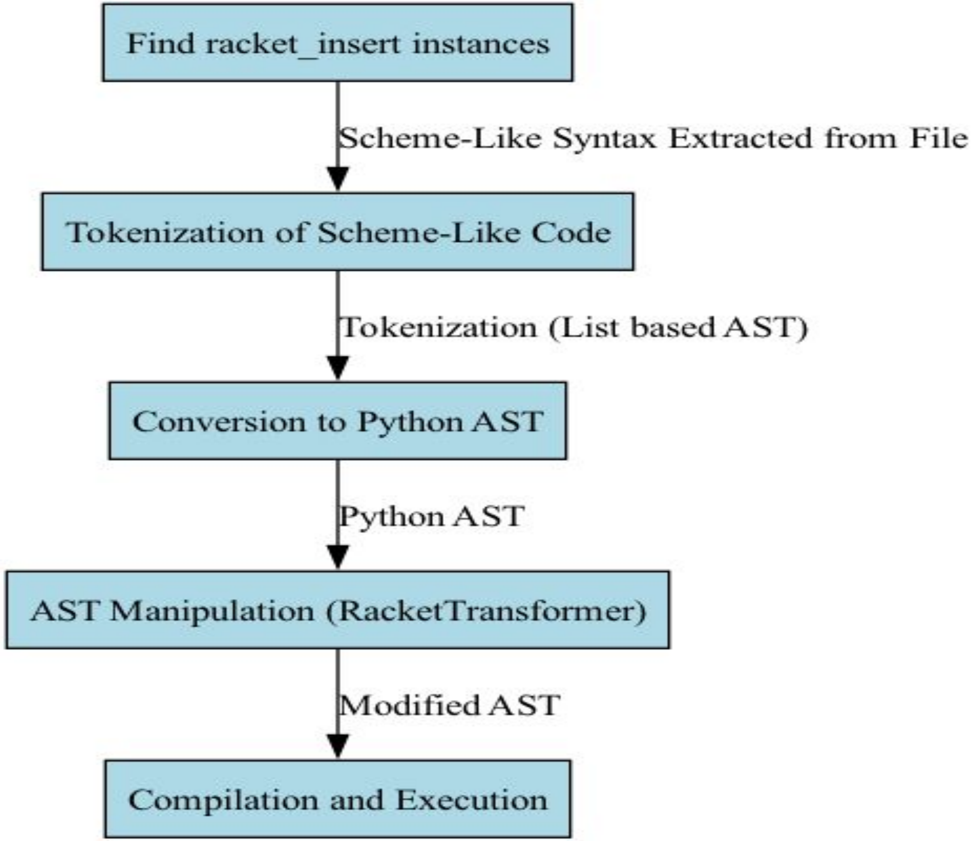
1. Identify racket_insert("file.rkt") in Python

2. Tokenize Scheme-like code from file

3. Convert tokens to Python AST

4. Dynamically replaces the racket_insert function call with the corresponding Python AST representation of the Scheme-Like code.

Pipeline Overview



Find racket_insert instances

Scheme-Like Syntax Extracted from File

Tokenization of Scheme-Like Code

Tokenization (List based AST)

Conversion to Python AST

Python AST

AST Manipulation (RacketTransformer)

Modified AST

Compilation and Execution

Similar Tools & Comparison

- Pybind11 : lightweight, header-only library specifically designed for exposing C++ types to Python and vice versa

- Cython : superset of the Python programming language that enables developers to write Python code with optional C-inspired syntax extensions

- Our Approach : Dynamic, runtime integration of Scheme-like code into Python

Possible Enhancements

- Ensure Robust error handling & type checks

- Performance optimizations

- Support more Racket features (macros, continuations)

THANK YOU!