CSC111 Lecture 11: Introduction to Abstract Syntax Trees

Hisbaan Noorani

February 22, 2021

Contents

1 Exercise 1: Representing assignment statements

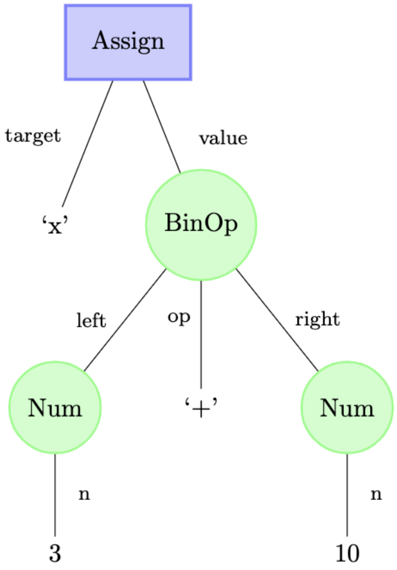
In lecture, we introduced the following class to represent assignment statements in abstract syntax trees.

```python
class Assign(statement):
    """An assignment statement (with a single target).

    Instance Attributes:
      - target: the variable name on the left-hand side of the equals sign
      - value: the expression on the right-hand side of the equals sign
    """
    target: str
    value: Expr

    def __init__(self, target: str, value: Expr) -> None:
        """Initialize a new Assign node."""
        self.target = target
        self.value = value
```

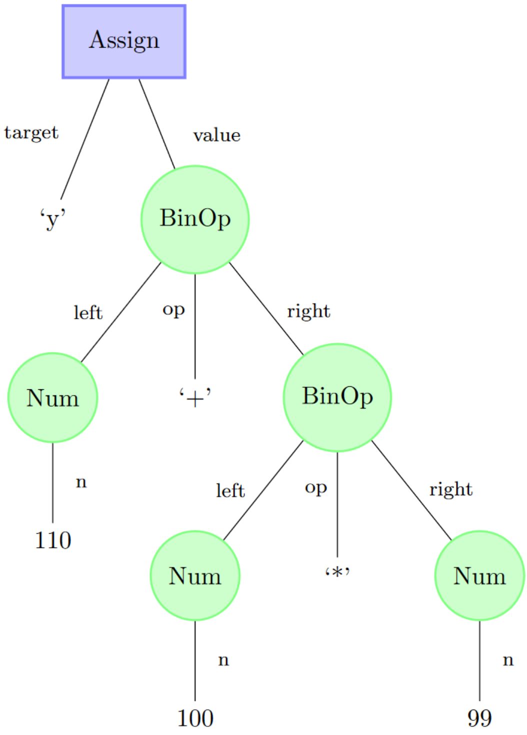
First, make sure you understand this class by answering the following questions.

1. Draw an abstract syntax tree diagram that represents the following Python statement.

```python
x = 10 + 3
```



2. Write an AST expression using `Assign` (and other AST types) to represent the following diagram.



```
Assign('y', BinOp(Num(10), '+', BinOp(Num(100), '*', Num(99)))
```

3. Finally, implement the `Assign.evaluate` method. This method should *mutate* its `env` argument, and shouldn't return anything.

```
class Assign(statement):
    def evaluate(self, env: dict[str, Any]) -> None:
        """Evaluate this statement.

        This does the following: evaluate the right-hand side expression,
        and then update <env> to store a binding between this statement's
        target and the corresponding value.

        >>> stmt = Assign('x', BinOp(Num(10), '+', Num(3)))
        >>> env = {}
        >>> stmt.evaluate(env)
        >>> env['x']
        13
        """
        env[self.target] = self.value.evaluate()
```

2 Additional exercises

1. Let's create a variation of the `Assign` class to support *parallel assignment*. Read through the following class.

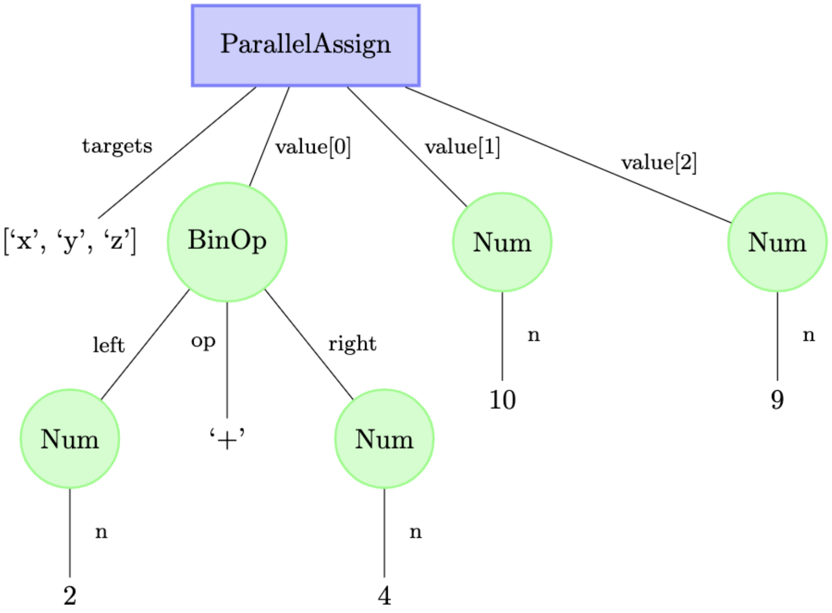
```
class ParallelAssign(Statement):
    """A parallel assignment statement.

    Instance Attributes:
        - targets: the variable names being assigned to---the left-hand side of the =
        - values: the expressions being assigned---the right-hand side of the =
    """
    targets: list[str]
    values: list[Expr]

    def __init__(self, targets: list[str], values: list[Expr]) -> None:
        """Initialize a new ParallelAssign node."""
        self.targets = targets
        self.values = values
```

To make sure you understand this class, answer the following questions.

(a) Draw the abstract syntax tree diagram that represents the following Python statement.

```python
ParallelAssign(['x', 'y'],
               [BinOp(Num(10), '+', Num(3)), Num(-4.5)])
```

(b) Write an AST expression using `Assign` (and other AST types) to represent the following diagram.



(c) Now, implement the `ParallelAssign.evaluate` method.

```python
class ParallelAssign:
    def evaluate(self, env: dict[str, Any]) -> None:
        """Evaluate this statement.

        This does the following: evaluate each expression on the right-hand side
        and then bind each target to its corresponding expression.

        Raise a ValueError if the lengths of self.targets and self.values are
        not equal.

        >>> stmt = ParallelAssign(['x', 'y'],
        ...                       [BinOp(Num(10), '+', Num(3)), Num(-4.5)])
        >>> env = {}
        >>> stmt.evaluate(env)
        >>> env['x']
        13
        >>> env['y']
        -4.5
        """
```