Semantics (VI)

Assignment Semantics (cont.)
- We now know that the meaning function of assignment can be expressed this mathematically,

\[ M(\text{Assignment}, \text{State}) = \text{state} \cup \{a . \text{target}, M(a . \text{source}, \text{state})\} \]

- Here, \( \cup \) means overriding union.
- If we have a set \( X \) which contains three pairs, \( \{<a, 1>, <b, 5>, <c, 1>\} \), and a set \( Y \) which contains two pairs, \( \{<b, 6>, <d, 0>\} \), the overriding union of \( X \) and \( Y \) is:

\[
\begin{align*}
X &= \{<a, 1>, <b, 5>, <c, -1>\} \\
Y &= \{<b, 6>, <d, 0>\} \\
X \cup Y &= \{<a, 1>, <b, 6>, <c, -1>, <d, 0>\}
\end{align*}
\]

The **overriding union** of \( X \) and \( Y \), written \( X \cup Y \), is the result of replacing in \( X \) all pairs \( (x, v) \) whose first member matches a pair \( (x, w) \) from \( Y \) by \( (x, w) \) and then adding to \( X \) any remaining pairs in \( Y \).

- Example
  - If \( X = \{<a, 1>\} \) and \( Y = \{<b, 2>\} \), what is the result of overriding union of \( X \) and \( Y \)?
    - \( \{<a, 1>, <b, 2>\} \)
  - If \( X = \{\} \) and \( Y = \{<c, 3>\} \)?
    - \( \{<c, 3>\} \)
  - If \( X = \{<d, 4>\} \) and \( Y = \{<d, 4>\} \)?
    - \( \{<d, 4>\} \)

- We are going to expand the simple interpreter we built for the expression and let it be able to determine the meaning of assignment.
- Add: Assignment class, Meaning function for assignment, and let the state can be printed out pretty. Also, a main function to test them.
# Assignment has the attribute target and source

class Assignment:
    def __init__(self, target, source):
        self.target = target
        self.source = source

# Maintain a dictionary that maps
# variables to their corresponding values

class State:
    def __init__(self):
        self.state = {}

    def setValue(self, var, value):
        self.state[var.name] = value

    def getValue(self, var):
        return self.state[var.name]

    def __str__(self):
        statestr = '{
            for var in self.state:
                statestr += '<' + var + ', "" + str(self.state[var]) + '>', '
        return statestr[:-2] + '}'

# Meaning function of Assignment
# M(Statement statement, State state)

def M_Assignment (statement, state):
    state.setValue(statement.target, M_Expression(statement.source, state).value)
    return state

def main2():
    # a = b + 3 x c {<a, 5>, <b, 1>, <c, 3>}
    vara = Variable('a')
    varb = Variable('b')
    varc = Variable('c')
    val3 = Value(3)
    expr = BinaryExpression(varb, '+', BinaryExpression(val3, '*', varc))
    assignment = Assignment(vara, expr)
    state = State()
    state.setValue(vara, 5)
    state.setValue(varb, 1)
    state.setValue(varc, 3)
    newState = M_Assignment(assignment, state)
    print(newState)

if __name__ == '__main__':
    #main1()
    main2()