Semantics (IV)

Side Effects

- *Note: Our discussion above assumes there are no side effects caused by evaluating the expressions.*
- A side effect occurs during the evaluation of an expression if, in addition to returning a value, the expression alters the state of the program.
- A typically example is the post- and pre- increment.
  - int `x = 1; int y = x++; int z = ++x;`
  - What are the x, y, and z's values after executing these three statements? [3, 1, 3]
- *If there are side effects, the above semantics are not specific enough.*
- The way to address side effects has two steps:
  1. Update the meaning function. Instead of producing a value, it returns a value and a state.

$$M : Expression \times State \rightarrow Value \times State$$

  2. The meaning rule should be more specified. Take the binary expression for instance, the meaning rule also define the order to determine the value of two terms.
     - If the Expression is a Binary, the meaning of $\mathrm{term1}$ in the current state is first determined, giving a value $\mathrm{v1}$ and a state $\mathrm{s1}$. Then the meaning of $\mathrm{term2}$ in state $\mathrm{s1}$ is determined, giving a value $\mathrm{v2}$ and a state $\mathrm{s2}$. Then **Meaning Rule of Binary Expression** determines the meaning of the expression by a*pplying the Operator* op *to the resulting values* $\mathrm{v1}$ *and* $\mathrm{v2}$ *in state* $\mathrm{s2}$, and the resulting state is $\mathrm{s2}$.
- *Note: We need to know the definition of side effect and the ways C and Java use to handle it. The implementation of side effects in meaning functions is out of the scope of the course. All the following implementations assume that there is no side effect.*

Assignment Semantics

- We know that the rhs of an assignment statement is an expression. To determine the meaning of an assignment statement, we need to know the meaning of the rhs expression.
- Now, we know how to determine the meaning of an expression. Then, let's figure out the way to determine the meaning of an assignment.

- We know that the meaning function of an expression generate a new value and the meaning function of an assignment generates a new state. (Remember: Assignment is a type of statement.)

$$M : Expression \times State \rightarrow Value$$
$$M : Statement \times State \rightarrow State$$

- This means the value generated by the rhs expression is used to update the value of the lhs variable of the assignment. In the new state, the lhs variable will be associated with the new value generated by the rhs expression. Expressed this mathematically,

$$M(Assignmenta, Statestate) = state \bar{\cup} \{a\,.target, M(a\,.source, state)\}$$

- Here, $\bar{\cup}$ 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:

$$X = \{\langle a, 1\rangle, \langle b, 5\rangle, \langle c, -1\rangle\}$$
$$Y = \{\langle b, 6\rangle, \langle d, 0\rangle\}$$
$$X \bar{\cup} Y = \{\langle a, 1\rangle, \langle b, 6\rangle, \langle c, -1\rangle, \langle d, 0\rangle\}$$

The *overriding union* of X and Y, written $X \bar{\cup} Y$, is the result of replacing in X all pairs $\langle x, v\rangle$ whose first member matches a pair $\langle x, w\rangle$ from Y by $\langle x, w\rangle$ 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>}
  - Given s = {<a, 2>, <b, 5>, <c, 4>}. Evaluate this assignment b = a + c. What is the new state?
    - {<a, 2>, <b, 6>, <c, 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):
        pairs = [
            f"<{name}, {value}>"
            for name, value in self.state.items()
        ]
        return "{" + ", ".join(pairs) + "}"



# 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()
```