

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. `x = 1; ++x + x++;`
- *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 term1 in the current state is first determined**, giving a value `v1` and a state `s1`. **Then the meaning of term2 in state `s1` is determined**, giving a value `v2` and a state `s2`. Then **Meaning Rule of Binary Expression** determines the meaning of the expression by *applying the Operator `op` to the resulting values `v1` and `v2` in state `s2`*, and the resulting state is `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.*
-

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,

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

- Here, $\bar{\cup}$ means overriding union.
- If we have a set X which contains three pairs, $\{ \langle a, 1 \rangle, \langle b, 5 \rangle, \langle c, 1 \rangle \}$, and a set Y which contains two pairs, $\{ \langle b, 6 \rangle, \langle d, 0 \rangle \}$, 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 $X = \{ \langle a, 1 \rangle \}$ and $Y = \{ \langle b, 2 \rangle \}$, what is the result of overriding union of X and Y ?
 - $\{ \langle a, 1 \rangle, \langle b, 2 \rangle \}$
 - If $X = \{ \}$ and $Y = \{ \langle c, 3 \rangle \}$?
 - $\{ \langle c, 3 \rangle \}$
 - If $X = \{ \langle d, 4 \rangle \}$ and $Y = \{ \langle d, 4 \rangle \}$?
 - $\{ \langle d, 4 \rangle \}$
- 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()

```

Conditional Semantics

- In addition to the assignment statement, conditional statement (if statement) is another type of statement.
- Its meaning function also takes in a state and generates another state.

$$M : \text{Statement} \times \text{State} \rightarrow \text{State}$$

- A conditional statement has three elements: an expression, then branch statement, and else branch statement.

$$\text{Conditional} = \text{Expression test}; \text{Statement thenbranch}, \text{elsebranch}$$

- The **meaning of a conditional depends on the meaning of the expression**.
 - If the expression is true, the meaning of the conditional is the meaning of the then branch.
 - If the expression is false, the meaning of the conditional is the meaning of the else branch.

$$M(\text{Conditional } c, \text{State } state) = \begin{cases} M(c.\text{thenbranch}, state), & \text{if } M(c.\text{test}, state) \text{ is true} \\ M(c.\text{elsebranch}, state), & \text{otherwise} \end{cases}$$