# Semantics (II)

## What is semantics? How do we define semantics?

- The semantics of a programming language is a precise definition of the meaning of any program that is syntactically and type-wise correct
- There are at least three ways to define the semantics
  - Operational semantics: the meaning of a program is the output produced by a given architecture/compiler pair (pre- and post- increment in C)
  - Axiomatize semantics: the meaning of a program can be rigorous proven by using a systematic logical argument (formal specification)
  - Denotational semantics: the meaning of a program can be described as a collection of **meaning functions** operating on the **program state** (this course)

## What is the program state?

- The state of a program is the binding of all variables to their current values.
- A variable and its value can be modeled as an ordered pair, e.g. `<i, 5>`
- A state is a set of ordered pairs (dictionary)

$$state = \{ < var1, val1 > , < var2, val2 > , \ldots < var_m, val_m > \}$$

- A program can have more than one states.
  - A state is a snapshot of the memory of the variables used by a program. After each statement execution, the program will have a new snapshot.

- Example
  - The program calculates factorial. Show the states of this program.

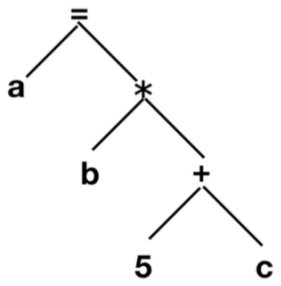
```
1 int main () {
2   int a, b;
3
4   a = 1;
5   b = 2;
6
7   b = a + b;
8
9   return 0;
10 }
```

Trace Table

| Snapshot | Before Statement | State |
|---|---|---|
| 1 | 4 | {<a, undef>, <b, undef>} |
| 2 | 5 | {<a, 1>, <b, undef>} |
| 3 | 7 | {<a, 1>, <b, 2>} |
| 4 | 9 | {<a, 1>, <b, 3>} |

## Meaning Function of Clite

- We know that the input of the semantics analysis stage of compilation is an abs syntax tree.
- So, to determine the meaning of a abs tree, we will figure out the precise meaning of each node/subtree of its abs tree.




- Consider an assignment statement. What do assignment statements do?
  - Modify the “state” of the program or the contents of the computer’s memory.
  - How do we indicate that mathematically?

    $$old\ state \rightarrow new\ state$$

  - This sounds like a function with input and output. An assignment statement is just a function that takes a state as input and outputs a state.

    ```
    M(assignment, oldstate) = newstate
    ```

    - M stands for “meaning”. The above function reads as “the meaning of this assignment statement in this state is this new state”

- An if statement changes the state as well. So use the same methodology, we can define the meaning function for if statement as

  ```
  M(ifstatement, oldstate) = newstate
  ```

- We can also use the same methodology to define the meaning function for a while statement.

  ```
  M(whilestatement, oldstate) = newstate
  ```

- Actually, all Clite statements are just functions from $state1 \rightarrow state2$
- We will combine all these functions into one function that takes two parameters.

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

- The meaning of a statement is a new state that depends both on the old state and the statement being executed.
- Follow the same methodology, we can define the function for Program and Expression:

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

- A value could be any kind of r-value such as an integer, float, or any other mathematically precise object.

- Let's start implementing M for Clite.
- Note that, since we are thinking of expressions and statements in terms of the abs tree, and since the meanings of each programming component (e.g., expression, assignment, statement, etc.) depend on the meaning of the subtrees, guess what we will have when we implement these functions? [Lots of recursion]

---

## Program Meaning Function

- The meaning function of Program can be implemented into this.

$$M : Program \rightarrow State$$
$$M(Program\ \text{p}) = M(\text{p.body}, InitialState(\text{p.decpart}))$$

- If we have a Program p, the implemented function should take two parameters, the body of p and the initial state of p's decpart.
- In this implementation, the InitialState is an auxiliary function that creates a state containing all the variables declared in p.decpart paired with their default values.
- For example, if `int x = 0, y = 0;` is p.decpart in Clite, then `initState(p.decpart) = {<x,0>, <y,0>}`.

- Note:
  - The "**meaning" of a Program** is the **final state of the computer's memory when the program halts**.
  - The p.body part uses recursion. Depends on the statement type in p.body, it will be implemented differently. For example, expression will have a different semantics than assignment.

## InitialState in Programming Languages

- The InitialState function must be clearly defined for a language in order for programs to have meaning. [show initialState.c, initialState.java, initialState2.java]
- C does not initialize the memory space. If not explicit initialization, the initial value of a variable is the random value stored in that memory space. (So, initialState.c gives different results every time you run it)

```c
/**
 * initialState.c
 * Ying Li
 */

#include <stdio.h>

int main () {
      int sum;

      for (int i = 0; i < 10; i++) {
        sum += 5;
      }

      printf("sum = %d \n", sum);

      return 0;
}
```

- Java requires the variables being initialized before using. So initialState.java gives an error for no initialization. But if the variables are fields in a class, even without a constructor, initialState2.java won't complain, since a default constructor will be called. Default constructor will initialize the variables to the default values of their types, e.g. int is 0, float is 0.0 and string is ""

```java
/**
 * InitialState.java
 * Ying Li
 */

public class InitialState {

      public static void main (String[] args) {
        int sum;

        for (int i = 0; i < 10; i++) {
                sum += 5;
        }

        System.out.printf("sum = %d \n", sum);
      }
}
```

```java
/**
 * InitialState2.java
 * Ying Li
 */

public class InitialState2 {

      private int sum;

      public int calSum () {
        for (int i = 0; i < 10; i++) {
                sum += 5;
        }

        return sum;
      }

      public static void main (String[] args) {

        InitialState2 s = new InitialState2();

        System.out.printf("sum = %d \n", s.calSum());
      }
}
```