

Introduction to Semantics

Overview

- So far, we have studied the syntax and have seen that we need both the grammar rules and some additional rules about how to parse the language so that we can generate abstract syntax trees that will enable us to generate the correct code. However, the trees alone aren't the code. We need to describe the step where we interpret the trees (either evaluating them as we go or generating machine code).
- We need to understand what each subtree means.

What are 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)
 - **Axiomatic semantics**: the **meaning** of a program **can be** rigorously **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)

$state = \{ \langle var1, val1 \rangle, \langle var2, val2 \rangle, \dots, \langle var_m, val_m \rangle \}$

- **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

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>}

```

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

```

Meaning Function of Clite

- We will start with the abstract syntax tree. We will figure out the precise meaning of each node of the 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.
- An if statement changes the state as well. All Clite statement are just functions from $state1 \rightarrow state2$
- We will combine all these functions into one function that takes two parameters.

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

$$\begin{aligned} M(\text{assignment}, \text{oldstate}) &= \text{newstate} \\ M(\text{ifstatement}, \text{oldstate}) &= \text{newstate} \\ M(\text{blockstatement}, \text{oldstate}) &= \text{newstate} \end{aligned}$$
- In each case, the new state depends both on the old state and the statement being executed.
- BTW, M stands for “meaning”, as in “the meaning of this assignment statement in this state is this new state”.
- In addition to this function, we need two more:

$$M : Program \rightarrow State$$

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

- 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 AST, and since the meanings of expressions and statements depend on the meaning of their 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.

$$M : Program \rightarrow State$$

$$M(Program\ p) = M(p.body, InitialState(p.decpart))$$

- 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 C , then $\text{initState}(p.\text{decpart}) = \{ \langle x, 0 \rangle, \langle y, 0 \rangle \}$.
- 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.
- 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)

```
/**
 * initialState.c
 * Ying Li
 * 10/06/2019
 */
#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 "".

```
/**
 * InitialState.java
 * Ying Li
 * 10/06/2019
 */
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);
    }
}
```

```
/**
 * InitialState2.java
 * Ying Li
 * 10/06/2019
 */
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());
    }
}
```