# Lenguajes de Programación

## Semántica

Ma. Laura Cobo

Universidad Nacional del Sur
Departamento de Ciencias e Ingeniería de la Computación
2018

# Semántica

La clase pasada, vimos como definir la sintaxis y la semántica estática de los lenguajes de programación.

La semántica dinámica trabaja con el significado de los constructores del lenguaje. No existe una notación universalmente aceptada de cómo expresar este tipo de semántica. Aunque se han desarrollado varios métodos

Los programadores obviamente necesitan conocer precisamente lo que las sentencias del lenguaje hacen. Pero generalmente lo infieren de las explicaciones en *inglés* que aparecen en los manuales de los lenguajes.

Los creadores de compiladores determinan la semántica del lenguaje para la que escriben el compilador nuevamente a partir de estas especificaciones en lenguaje natural.

2

# Semántica

Se han desarrollado varios sistemas notacionales para la definición formal de la semántica que están incrementalmente en uso:

- *A través del manual de referencia de lenguaje*: este es el método más común, pero adolece de falta de precisión asociada a los defectos del uso de lenguaje natural. Terminan resultando inadecuadas por estar basadas en técnicas de implementación e intuición.
- *A través del traductor*: es el método mas común para cuestionar el comportamiento de un lenguaje en forma interactiva. Es esta aproximación, el programa es ejecutado, para determinar su comportamiento. Su principal desventaja, radica en que el traductor depende la máquina y puede no ser portable a todas.
- *A través de una definición formal*: es el método mas común para cuestionar el comportamiento del programa a través de métodos matemáticos, que son precisos pero también complejos y abstractos.
  Su ventaja es que son tan precisos, que pueden probar su validez matemática y la traducción puede ser verificada de manera de asegurar el comportamiento exacto de la definición.

3

# Semántica

Métodos de prueba

**Semántica operacional**: describe el significado de una programa ejecutando sus sentencias en una máquina (simulada o real). De esta manera el cambio de esta de la máquina (cambio en su memoria, registros, etc) define el significado de la sentencia

**Semántica axiomatica**: está basada en lógica formal. Los axiomas o reglas de inferencia están definidos para cada tipo de sentencia en el lenguaje, transformando expresiones en otras expresiones (llamadas aserciones).

Originalmente fue pensada para la verificación formal de programas.

**Semántica denotacional**: está basada en la teoría de funciones recursivas.

Éste método es el más abstracto de los métodos para describir semántica

Model theory

4

# Semántica

Los métodos formales resultan importantes por varias razones:

- Proveen una manera no ambigua de definir el lenguaje.
- Proveen estándares de manera que el lenguaje no sufra variaciones de implementación en implementación.
- Proveen las bases para pruebas de correctitud tanto de los compiladores como de los programas.

Los métodos que presentaremos son apropiados para la definición de la semántica de lenguajes imperativos (los tres presentados en la trasparencia anterior).

5

# Semántica axiomática: Introducción

En semántica axiomática se utilizan elementos de la lógica matemática para especificar las semántica de un lenguaje de programación y sus componentes.

El primer aporte significativo en esta aproximación, fue realizado por Hoare y Wirth en 1980. más precisamente para proveer la semántica del lenguaje Pascal.

Con esta semántica, se describe el significado de cada programa sintácticamente correcto asociando cada constructor las propiedades que tienen las variables antes de que la ejecución comience y luego de que el programa termina o el lo hace el constructor. Básicamente se utilizan fórmulas de correctitud, es decir, se utilizan predicados o aserciones.

6

# Semántica axiomática: Introducción

El cálculo de predicados, se utiliza como metalenguaje de la semántica axiomática a fin de expresar las suposiciones iniciales y los resultados esperados de la ejecución.

Fórmula Pre-post condición: {P} *sentencia* {Q}

Ejemplo:

a= b+1 {a > 1}

Posible precondición: {b > 10}

Precondición más débil: {b > 0}

7

# Semántica axiomática: proceso de prueba

La idea es determinar la pos-condición del programa, es decir expresar cuál es el resultado esperado.

Luego ir retrocediendo el programa hasta alcanzar la primera sentencia. Si la precondición de la primera sentencia es misma que la de la especificación del programa, entonces el programa es correcto.

8

# Semántica axiomática: axiomas

La semántica determina axiomas para los constructores. El axioma para los ciclos repetitivos es inherentemente más complejo debido a que el número de iteraciones no puede ser determinado en todos los casos. Cuando la cantidad de iteraciones es conocida, el ciclo puede tratarse como una secuencia.

El problema de computar la precondición más débil de un ciclo es similar a las demostraciones para enteros positivos, implica una inducción. Para ello debemos plantear la hipótesis, es este caso el *invariante de ciclo*. Su determinación es crucial para encontrar la precondición más débil.

$$\frac{\{I \text{ and } B\}\ S\ \{I\}}{\{I\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{end}\ \{I \text{ and } (\text{not } B)\}}$$

9

# Semántica axiomática: axiomas

*Ciclos tipo while (continuación)*:

Para encontrar el *invariante de ciclo*. Se puede utilizar un método similar al usado para determinar la hipótesis inductiva en una inducción matemática.

El invariante de ciclo debe verificar:

1. La precondición del ciclo debe garantizar la veracidad del invariante.
2. El invariante debe garantizar la veracidad de la pos-condición cuando termina el ciclo.
3. Durante la ejecución del ciclo, el valor del verdad del invariante no debe verse afectado; es decir la evaluación del control del ciclo no debe cambiar su valor de verdad

Otro aspecto complicado es la terminación, si se garantiza Q como pos-condición del ciclo, entonces la precondición P del mismo debe garantizar Q al terminar el ciclo como así también su terminación.

10

# Semántica axiomática: axiomas

*Ciclos tipo while (continuación)*:

Si I es el *invariante de ciclo* se requieren los siguientes axiomas para el ciclo:

1. P => I
2. {I} B {I}
3. {I and B} S {I}
4. (I and (not B)) => Q
5. El ciclo termina

11

# Semántica axiomática: evaluación

1. Obtener axiomas para todas las sentencias del lenguaje no es una tarea sencilla.
2. Es una excelente herramienta para determinar la correctitud de programas y un framework excelente para razonar sobre el mismo.
3. No es muy útil para los programadores y para los diseñadores de compiladores.
4. Su utilidad para describir el significado de los programas queda limitada por los programadores o quienes escriben los compiladores.

12

# Semántica operacional: Introducción

La *semántica operacional* define el comportamiento de una lenguaje de programación describiendo sus acciones en término de operaciones sobre una máquina abstracta o hipotética.

Requiere que las operaciones de la máquina utilizada para definición semántica esté definida en forma precisa.

Establece la forma de expresar semántica a más bajo nivel, ya que la misma está definida por la arquitectura de la máquina, sin embargo esta aproximación tiene aspectos que no la hacen totalmente apropiada para la especificación semántica:

1. Las acciones pueden ser muy difíciles de entender por complejidades del hardware o sistema operativo.
2. La especificación semántica dada, solo sirve para sistemas de cómputo idénticos.
3. Para lenguajes complejos, es difícil escribir intérpretes correctos.

13

# Semántica operacional

Algunos problemas pueden evitarse sustituyendo la máquina abstracta por simulaciones sobre una computadora real. Se simula con una cantidad discreta de estados y secuencias explícitas de operaciones de cómputo.

- El proceso
  - Construir el traductor (traduce el código fuente al código máquina de la computadora ideal)
  - Construir un simulador para la computadora ideal.
- Evaluación de la semántica operacional
  - Buena si es utilizada en forma informal (manuales del lenguaje, etc)
  - Extremadamente compleja si es utilizada en forma formal (ejemplo VDL), fue utilizada para describir la semántica de PL/I.

14

# Semántica denotacional: Introducción

La *semántica denotacional* (o semántica matemática) fue desarrollada a principios de la década del 70 por Christopher Strachey y Dana Scott sobre bases puramente matemáticas.

En esta aproximación, los programas pueden ser traducidos a funciones y sus propiedades probadas utilizando la teoría de funciones matemática estándar, conocida como *cálculo funcional*.

Originalmente se desarrollo para el análisis de lenguajes de programación; sin embargo se convirtió en una herramienta poderosa para el diseño e implementación de lenguajes.

15

# Semántica denotacional: principios

La *semántica denotacional* define el comportamiento de un lenguaje de programación aplicando funciones matemáticas a programas para representar su significado.

Una ventaja de la semántica denotacional es que se puede predecir el comportamiento de un programa sin necesidad de correrlo en una computadora.

16

# Semántica denotacional: ciclos repetitivos

El significado del ciclo es el valor de las variables del programa luego de que las sentencias del ciclo han sido ejecutadas el número previsto de veces, asumiendo que no hubo errores.

En esencia, el ciclo a sido convertido de interativo a recursivo, donde el control recursivo es matemática, definido por otras funciones de mapeo recursivo de estados.

Esto es bueno porque la recursión puede ser descripta más fácilmente con rigor matemático que la iteración

17

# Semántica denotacional: evaluación

- Puede ser utilizado par aprobar la correctitud de programas.
- Provee una forma rigurosa de pensar sobre programas.
- Puede ser una ayuda en el diseño de lenguajes.
- Ha sido utilizado en sistemas de generación de compiladores.
- Debido a su complejidad, son de poco uso por parte de los usuarios de los lenguajes de programación.

18