Principios de Lenguajes de Programación
Clase 4 - Descripción Semántica

Sandra Roger

Facultad de Informática
Universidad Nacional del Comahue

Primer Cuatrimestre

Unidad 1. *Introducción a los conceptos de los lenguajes de programación*

Concepto del del lenguaje de programación. Buenos Lenguajes y Atributos. Historia y Evolución de los Lenguajes de Programación. Clasificación de lenguajes. Criterios de diseño y de implementación de un lenguaje de programación. Concepto de Paradigma: imperativo, funcional, lógico, orientado a objetos. Máquinas virtuales: Jerarquías.
Sintaxis y semántica. Conceptos de Intérpretes y Compiladores. Técnicas formales de descripción sintáctica. Introducción a la Semántica Operacional, Axiomática y Denotacional. Atributos y Tiempos de Ligadura.

Por ejemplo

“existen 10 tipos de personas:
las que entienden los números binarios
y las que no”

Índice

- Introducción
- Semántica formal
- Semántica Operacional

¿Qúe vimos?

- Sintaxis: estructura o forma de los programas
- Semántica: relac. entre un pgm y un modelo de computación
- Semántica Estática: considera los aspectos que no pueden ser capturados por la sintaxis (debido a dificultad o imposibilidad de la gramática) pero que pueden ser evaluados en compilación
- Semántica Dinámica: aquellas propiedades que describen la interpretación de las frases significativas del lenguaje y tales elementos del lenguaje pueden realizarse con la ejecución de una máquina abstracta, una vez traducido el programa, ejemplos, los cambios de valores de las variables
- Pragmática: grado de éxito con el que un programa cumple sus objetivos tanto en su fidelidad con el modelo de computación subyacente como su utilidad para los programadores

Semántica y Sintaxis

- Por ejemplo:
  - Operación de "Asignación"

```
Raiz1 := -B + sqrt((b^2-4*A*C))/(2*A)
```

    - Sintaxis
      - Árbol Sintáctica - ¿Ambiguo?
    - Semántica
      - Semántica Estática
      - Semántica Dinámica

Semántica

- Semántica:
  - históricamente informal → descriptivo
  - explicación de los programas y hasta los lenguajes (texto en prosa)
- Recientemente: necesidad de descripción más formal (matemática)
  - Precisión en la definición
    - se puede demostrar la correctitud de un programa (en base a la matemática)
  - Validación de compiladores (traductores):
    - comportamiento exacto a la "definición" del lenguaje

Semántica

- 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.

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.

Semántica

- ¿Porqué tiene que ser formal?
  - 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 programadores / constructores de compiladores pueden conocer "exactamente" lo que hace (o debe hacer) un lenguaje

Semántica

- Problema de las Semánticas Formales:
  - Los lenguajes actuales son muy complejos
  - Área de investigación
- *Pocos* lenguajes fueron descriptos formalmente
  - Por ejemplo *Scheme*

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

Semántica Formal

- Métodos desarrollados
  - *Semántica Denotacional:*
    - Uso de funciones para especificar semántica, los programas se convierten en funciones para poder aplicar la teoría de funciones matemática
  - *Semántica Axiomática:*
    - Aplica la lógica formal: afirmaciones para describir suposiciones y resultados deseados
    - Los axiomas o reglas de inferencias son usados en cada tipo de sentencias(permite *transformaciones de expresiones a otras expresiones)*

Semántica Operacional

- Describe el significado (semántica) de un programa ejecutando las **acciones** del programa en una **máquina** (computadora), real o hipotética (simulada)
- Los cambios en el **estado** de la máquina (su memoria, registros, etc.) definen el significado de las sentencias del programa.
- Requiere precisión en la definición de las acciones de la "máquina", y muchas veces la máquina se parece muy poco a una computadora real

Semántica Operacional

- En lenguaje de alto nivel, es necesario definir una "*máquina virtual*".
- Un intérprete puro en *hardware* sería muy costoso
- Un intérprete puro en *software* normalmente tiene problemas
  - Las características detalladas de un computadora existente haría las acciones muy complicadas de comprender
  - Por ello, la definición de la semántica sería dependiente de la máquina

Semántica Operacional

- Otra alternativa mejor: una ***simulación completa*** de una computadora
- Proceso
  - Construir un compilador (traductor) que traduce el código fuente al código máquina de la computadora ideada
  - Construir un compilador para la máquina ideada
- Evaluación de la semántica operacional
  - Es buena cuando se usa informalmente (definir manuales de lenguajes, etc.)
  - Extremadamente compleja cuando se usa formalmente

Semántica Operacional

- Definición de máquina abstracta
  - Programa <--> Control <--> Almacenamiento
  - SO: como reacciona la máquina en un lenguaje: cómo cambia el almacenamiento durante la ejecución
- Base: máquina de reducción
  - Reducir un programa a su valor semántico
  - Ejemplo

```
(3+4)*5  =>  (7)*5
         =>  7*5
         =>  35
```

Reglas de Inferencia lógica

- De la forma:

$$\frac{premisa}{conclusión}$$

- por ejemplo, la propiedad transitiva:

$$\frac{a \rightarrow b, b \rightarrow c}{a \rightarrow c}$$

Semántica Operacional: aplicación

- Notación regla de inferencia lógica para definir el control reduce las construcciones (ej expresión) a su valor.
- Para explicarla usamos una parte de un lenguaje (expresiones)

$$E \rightarrow E_1\text{'+'}E_2 \mid E_1\text{'-'}E_2 \mid E_1\text{'*'}E_2 \mid \text{'('}\ E_1\ \text{')'}$$
$$N \rightarrow N_1 D \mid D$$
$$D \rightarrow \text{'0'} \mid \text{'1'} \mid \text{'2'} \mid \ldots \mid \text{'9'}$$

- Estilo: *semántica operacional estructural*

Semántica operacional: reglas

- *Regla 1: reducir dígitos a valores (axiomas)*

  $'0' \Rightarrow 0$
  $'1' \Rightarrow 1$
  $'2' \Rightarrow 2$
  ...
  $'8' \Rightarrow 8$
  $'9' \Rightarrow 9$

- *Regla 2: reducir números a valores (axiomas)*

  $V'0' \Rightarrow 10*V$
  $V'1' \Rightarrow 10*V + 1$
  $V'2' \Rightarrow 10*V + 2$
  ...
  $V'9' \Rightarrow 10*V + 9$

Semántica operacional: reglas

- Regla 3: $V_1\ '+'\ V_2 => V_1 + V_2$
- Regla 4: $V_1\ '-'\ V_2 => V_1 - V_2$
- Regla 5: $V_1\ '\star'\ V_2 => V_1 \times V_2$
- Regla 6: $'('\ V\ ')' => V$
- Regla 7: $$\frac{E => E_1}{E\ '+'\ E_2 => E_1\ '+'\ E_2}$$
- Regla 8: $$\frac{E => E_1}{E\ '-'\ E_2 => E_1\ '-'\ E_2}$$
- Regla 9: $$\frac{E => E_1}{E\ '\star'\ E_2 => E_1\ '\star'\ E_2}$$

Semántica operacional: reglas

- *Regla 10:* $$\frac{E => E_1}{V\,'+'\,E => V\,'+'\,E_1}$$
- *Regla 11:* $$\frac{E => E_1}{V\,'-'\,E => V\,'-'\,E_1}$$
- *Regla 12:* $$\frac{E => E_1}{V\,'*'\,E => V\,'*'\,E_1}$$
- *Regla 13:* $$\frac{E => E_1}{'('\,E\,')' => '('\,E_1\,')'}$$
- *Regla 14:* $$\frac{E => E_1\,,\; E_1 => E_2}{E => E_2}$$

Semántica operacional: aplicación

- Evaluamos la expresión del lenguaje

  *2 * (3 + 4) - 5*

- La expresamos y reducimos

  '2' '' '(' '3' '+' '4' ')' '-' '5'*

- Tomamos

  '3' '+' '4' => ***3*** *'+' '4'* (reglas 1 y 7)
  => ***3*** *'+'* ***4***
  => ***3 + 4*** (reglas 3)
  => *7*

- Luego

  '2' '' '(' '3' '+' '4' ')'* => ***2*** *'*' '(' '3' '+' '4' ')'* (reglas 1 y 9)
  => ***2*** *'*' *7* (regla 12)
  => ***2 × 7 = 14*** (regla 5)

Semántica operacional: aplicación

- Finalmente

  '2' '' '(' '3' '+' '4' ')' '-' '5'* => ***14*** *'-' '5'* (r. 1 y 8)

  => ***14 '-' 5*** (r. 11)

  => ***14 - 5*** (r. 4)

  => ***9***

- Demostrando que

  *2 * (3 + 4) - 5*

  Se reduce a

  9

Semántica Operacional: cont.

- Extendemos a ambientes y asignaciones

$$P \rightarrow L$$
$$L \rightarrow L_1 ';' L_2 \mid S$$
$$S \rightarrow I ':=' E$$
$$E \rightarrow E_1 '+' E_2 \mid E_1 '-' E_2 \mid E_1 '*' E_2 \mid '(' E_1 ')' \mid N$$
$$N \rightarrow N_1 D \mid D$$
$$D \rightarrow '0' \mid '1' \mid '2' \mid \ldots \mid '9'$$
$$I \rightarrow I_1 A \mid A$$
$$A \rightarrow 'a' \mid 'b' \mid 'c' \mid \ldots \mid 'z'$$

- Incorporamos el ambiente ***Env*** para incluir el efecto en la máquina abstracta, usando la notación < *E* | *Env* > para indicar la expresión *E* es evaluada en el entorno ***Env***

Semántica operacional: cont. reglas

- *La Regla 7:* $$\frac{E => E_1}{E\ '+'\ E_2 => E_1\ '+'\ E_2}$$
- Se modifica para transformarse en:

$$\frac{<E \mid Env> => <E_1 \mid Env>}{<E\ '+'\ E_2 \mid Env> => <E_1\ '+'\ E_2 \mid Env>}$$

Semántica operacional: cont. reglas

- *Regla 15:* $$\frac{Env\ (I) => V}{< I \mid Env > \ => < V \mid Env >}$$
- *Regla 16:* $< I\ ':='\ V \mid Env > \ => Env\ \&\ \{ I = V \}$
- *Regla 17:* $$\frac{<E \mid Env > => <E_1 \mid Env >}{< I\ ':='\ E \mid Env > => \ < I\ ':='\ E_1 \mid Env >}$$
- *Regla 18:* $$\frac{<S \mid Env > => \ Env_1}{<S\ ';'\ L \mid Env > => \ < L \mid Env_1 >}$$
- *Regla 19:* $L \ => < L \mid Env_0 >$

  con $Env_0(I)$ = indefinido para todos los identificadores I

Semántica operacional: aplicación

- Evaluamos el siguiente programa en el lenguaje

  a := 2 + 3;

  *b := a * 4;*

  a := b – 5;

Semántica Operacional: cont.2

- Extendemos a las sentencias de control

$$S \rightarrow \text{'if'}\ E\ \text{'then'}\ L_1\ \text{'else'}\ L_2\ \text{'fi'}\ \mid$$
$$\text{'while'}\ E\ \text{'do'}\ L\ \text{'od'}$$

Semántica operacional: aplicación

- Evaluamos el siguiente programa en el lenguaje

  a := 2 + 3;

  *b := a * 4;*

  a := b – 5;

Semántica operacional: cont.2 r.

- *Regla 20:*

$$\frac{<E \mid Env> \Rightarrow <E_1 \mid Env>}{<\text{if' } E \text{ 'then' } L_1 \text{ 'else' } L_2 \text{ 'fi'} \mid Env> \Rightarrow <\text{if' } E_1 \text{ 'then' } L_1 \text{ 'else' } L_2 \text{ 'fi'} \mid Env>}$$

- *Regla 21:*

$$\frac{V > 0}{<\text{if' } V \text{ 'then' } L_1 \text{ 'else' } L_2 \text{ 'fi'} \mid Env> \Rightarrow <L_1 \mid Env>}$$

- *Regla 22:*

$$\frac{V \leq 0}{<\text{if' } V \text{ 'then' } L_1 \text{ 'else' } L_2 \text{ 'fi'} \mid Env> \Rightarrow <L_2 \mid Env>}$$

Semántica operacional: cont.2 r.

- *Regla 23:*

$$\frac{<E \mid Env> => <V \mid Env>, \quad V \leq 0}{<\text{'while'}\ E\ \text{'do'}\ L\ \text{'od'} \mid Env> => Env}$$

- *Regla 24:*

$$\frac{<E \mid Env> => <V \mid Env>, \quad V > 0}{<\text{'while'}\ E\ \text{'do'}\ L\ \text{'od'} \mid Env> => <L;\text{'while'}\ E\ \text{'do'}\ L\ \text{'od'} \mid Env>}$$

Semántica operacional: aplic.2

- Evaluamos el siguiente programa en el lenguaje

```
n := 0 – 3 ;
if  n
   then  i := n
   else   i := 0 – n
fi ;
fact := 1 ;
while  i  do
     fact := fact * i ;
     i := i - 1 ;
od
```

Semántica operacional: aplic.2

- Desafío: Resolver
- Ayuda: ambiente al comienzo del ciclo es:

  $\{ n = -3, i = 3, fact = 1 \}$

  Aplica 24 y luego por regla 18 se calcula el ambiente resultante de aplicar el cuerpo con el ambiente inicial

Semántica Axiomática

- Propósito original: verificación formal de programas
- Los axiomas o reglas de inferencias se definen para cada tipo de instrucción para permitir transformaciones de las expresiones en otras expresiones
- Las expresiones se llaman *aserciones (assertions)*

Semántica Axiomática

- En semántica axiomática se utilizan elementos de la lógica matemática para especificar las semántica de un lenguaje de programación y sus componentes.
- Se usan "axiomas" lógico y matemáticos para especificar la semántica los LdP

Semántica Axiomática (cont)

- Una aserción anterior a una sentencia (***precondición***) establece las relaciones y restricciones entre las variables que son verdaderas en ese punto de ejecución
- Una aserción que sigue a una sentencia es una ***postcondición***
- **La *precondición más débil*** (weakest precondition) es la condición menos restrictiva que garantiza la postcondición en ese programa

Forma de Semántica Axiomática

- Especificación axiomática de la semántica

  $\{ P \} \quad C \quad \{ Q \}$

  - P y Q afirmaciones y C conjunto de sentencias
- Ejemplo

  a = b + 1

  {a > 1}
- Precondiciones:
  - Una posible: {b > 10}
  - La más débil (WP): {b > 0}

Forma de Semántica Axiomática

- 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.

Semántica Axiomática: evaluación

- El desarrollo de los axiomas y las reglas de inferencia para todas las sentencias de un lenguaje es complejo (dificultoso)
- Una buena herramienta para demostrar correctitud, y un entorno excelente para razonar sobre los programas
- No es muy útil para los usuarios de lenguajes y los implementadores de compiladores
- Su utilidad resulta limitada por su dificultad de uso, usado sólo un número limitado de usuarios del lenguaje y de los desarrolladores de herramientas.

Semántica Denotacional

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.

Semántica Denotacional

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.

Bibliografía

- Louden (material en pedco)
- Sebesta, Robert, Concepts of Programming Languages, Cap 3
- Pratt, Terrance W., Programming Languages: Design and Implementation, cap 9