Semántica Axiomática

Semántica axiomática

- Basado en lógica matemática (cálculo de predicados)
- Propósito original: Verificación formal del programa
- Axiomas o reglas de inferencia son definidas para cada tipo de sentencia del lenguaje (a fin de permitir transformaciones de expresiones en expresiones de lógica más formales)
- Las expresiones lógicas son denominan *aserciones*
  - Basadas en el cálculo de predicados
  - No representan el estado completo del programa, apenas lo que será alterado por una sentencia

Semántica axiomática

- Una aserción que precede inmediatamente una sentencia (una pre-condición) describe las restricciones en las variables del programa en aquel punto.
- Una aserción inmediatamente después de una sentencia es una pos-condición.
- Ejemplo de pre-condición, sentencia y pos-condición:
  - **{x ≥ 0} sum = 2 * x + 1 {sum ≥ 1}**
- Una pre-condición más debil es menos restrictiva que garantizará la validez de la pos-condición asociada
  - En el ejemplo arriba {x ≥ 10}, {x ≥ 100} no invalidan {sum ≥ 1}
  - Sin embargo, restringen el valor de x más de lo necesario
  - La precondición más debil es mismo {x ≥ 0}

Forma de la semántica axiomática

- Forma sentencial: `{P} statement {Q}`
- Un ejemplo más
  - `a = b+1 {a > 1}`
  - Una pre-condición posible: `{b > 10}`
  - Pre-condición más debil: `{b > 0}`

Pruebas de programas

- En general, ud. informa cual es el resultado esperado del programa
  - Sería la pos-condición final del programa como un todo
  - Regrese hasta la primera sentencia del programa. Si la pre-condición en la primera sentencia es la misma de la especificación del programa, este está correcto.
  - Ese proceso de camino inverso responde a la pregunta: Para cuales valores de entrada el valor deseado es obtenido?
- Tener seguridad de cuales valores producen la salida esperada evita entregar entrada que puede causar ejecuciones imprevisibles del programa

Pruebas de programas

- Un axioma para sentencia de asignación

  (x = E): $\{Q_{x \to E}\}$ x = E {Q}

  - *x* = *E*, *x* es una variable y *E* es una expresión, luego *x*=*E* es la sentencia en cuestión.
  - Q es la pos-condición. En general, es informada, como algo que se desea garantizar.
  - $Q_{x \to E}$ seria la pre-condición, que es obtenida substituyendo x por E en Q
- Considere las siguientes sentencias y pos-condiciones:
  - `a = b / 2 - 1 {a < 10}`
  - `x = 2 * y - 3 {x > 25}`
- ¿Cuales serian las pre-condiciones por el axioma de asignación?

Reglas de inferencia

- Una regla de inferencia es un método de inferir la verdad de una aserción con base en los valores de otras aserciones: $\frac{S1, S2, \ldots, Sn}{S}$
- Si S1, S2, …, y Sn fuesen verdaderas, S es verdadera.
- La Regla de Consecuencia

$$\frac{\{P\}\ S\ \{Q\},\ \ P' \Rightarrow P,\ \ Q \Rightarrow Q'}{\{P'\}S\{Q'\}}$$

  - La regla de consecuencia nos permite considerar la pre-condición más debil.
    - { x>3 } x = x-3 { x>0 } está correcto
    - Pero si en lugar de { x>3 }, usasemos { x>5 }, ¿estariamos equivocados?

Semántica axiomática: secuencias

- Para la semántica axiomática, podemos considerar una regla de inferencia para secuencias de la forma S1;S2 con
- {P1} S1 {P2}
- {P2} S2 {P3}
- De la forma:

$$\frac{\{P1\}\ S1\ \{P2\},\ \{P2\}\ S2\ \{P3\}}{\{P1\}\ S1;\ S2\ \{P3\}}$$

- Caso *S1: y = 3 * x + 1;* y *S2: x = y + 3;* , teniendo como pos-condición { x < 10 }, ¿cuál sería la pre-condición?

Semántica axiomática: Selección

- Considere el siguiente formato para una estructura de selección:
  - **if** B **then** S1 **else** S2
- Tenemos la siguiente regla de inferencia:

$$\frac{\{\text{B and P}\}\ \text{S1}\ \{\text{Q}\},\ \{(\text{not B}) \text{ and P}\}\ \text{S2}\ \{\text{Q}\}}{\{\text{P}\}\ \texttt{if}\ \text{B}\ \texttt{then}\ \text{S1}\ \texttt{else}\ \text{S2}\ \{\text{Q}\}}$$

- Para la sentencia y pos-condición abajo, ¿cual es la precondición?

if x > 0 **then** y = y - 1 **else** y = y + 1 {y > 0}

Semántica axiomática: lazo *while*

- Una regla de inferencia para el lazo de repetición

  {P} **while** B **do** S **end** {Q}

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

  donde I es la invariante del lazo de repetición

Semántica axiomática: lazo *while*

- Características de la invariante del lazo de repetición: I debe satisfacer requisitos:
  - P => I -- la invariante de repetición precisa ser verdadera inicialmente
  - {I and B} S {I} -- evaluación de B no puede cambiar la validez de I
  - (I and (not B)) => Q -- I no es cambiado por la ejecución del cuerpo de repetición
  - La repetición termina -- puede ser difícil de probar

Semántica axiomática: lazo *while*

- La invariante de repetición es una versión debilitada de la pos-condición de repetición y es también una pre-condición.
- I debe ser débil lo suficiente para estar llenado antes de l inicio del ciclo, pero cuando fuese combinado con la condición de salida del lazo debe ser suficientemente fuerte para forzar la verdad de la pos-condición
- Transformador de predicado:
  - *wp* (sentencia, pos-condición) = pre-condición
  - Es el proceso que ya hicimos en las otras reglas de inferencia
- Considerando los lazos y pos-condiciones:
  - while y <> x do y = y + 1 end {y = x}
  - while s > 1 do s = s / 2 end {s = 1}
- ¿Cuáles serian invariantes

Semántica axiomática: lazo *while*

- La invariante de repetición es una versión debilitada de la pos-condición de repetición y es también una pre-condición.
- I debe ser débil lo suficiente para estar llenado antes de l inicio del ciclo, pero cuando fuese combinado con la condición de salida del lazo debe ser suficientemente fuerte para forzar la verdad de la pos-condición
- Transformador de predicado:
  - *wp* (sentencia, pos-condición) = pre-condición
  - Es el proceso que ya hicimos en las otras reglas de inferencia
- Considerando los lazos y pos-condiciones:
  - while y <> x do y = y + 1 end {y = x}
  - while s > 1 do s = s / 2 end {s = 1}
- ¿Cuáles serian invariantes

Semántica axiomática: prueba de programas

- Quiero probar que la siguiente descripción está correcta

```
{n >= 0}
count = n;
fact = 1;
while count <> 0 do
     fact = fact * count;
     count = count - 1;
end
{fact = n!}
```