Universdad Nacional Autónoma de México

Lenguajes de Programacion
Examen Parcial III

- Edgar Montiel Ledesma
  317317794
- Carlos Daniel Cortes Jimenez
  420004846

---

1. Problema

El constructor alternativo if se puede generalizar mediante un operador case definido como sigue:

e ::= . . . — case g end

g ::= e1 ⇒ e2 — g ; g

Una expresion de la forma e1 ⇒ e2 se conoce como expresion resguardada, siendo la expresion e1 una expresion booleana llamada guardia. Un constructor case se evalua como sigue: recorrer las expresiones resguardadas ei ⇒ ej en orden de izquierda a derecha (respectivamente de arriba a abajo), hasta hallar la primera expresion resguardada, digamos ek ⇒ el tal que ek ⇒ ∗ true, en cuyo caso se procede a evaluar el, cuyo valor final es tambien el resultado de la evaluacion de la expresion case. Por ejemplo considerese el siguiente programa:

case x=0 =¿ x ;

x¿2 =¿ x^2 ;

x¿0 =¿ x-2 ;

x¡0 =¿ x+2

end

2. Preguntas

1. Define la sintaxis abstracta del operador case.

   La sintaxis abstracta para el operador case en la siguiente:

   $$\frac{t\ asa}{case(t,g)\ asa}$$

   Donde:

   - **t** es la expresión que se evaluará para determinar el valor a considerar en las expresiones resguardadas.

- **g** es una secuencia de expresiones resguardadas y sus respectivos resultados, de la forma $e_1 \Rightarrow e_2$.

2. Define las reglas de transicion para modelar la semantica operacional del nuevo operador case.

$$\frac{}{case(a,g) \rightarrow a_k} \quad \text{case}$$

Donde:

- **a** es la expresión que se evaluará para determinar el valor a considerar en las expresiones resguardadas.
- **g** es una secuencia de expresiones resguardadas y sus respectivos resultados, de la forma $e_1 \Rightarrow e_2$.

3. Define las reglas de tipado para la semantica estatica del operador case.

   En cada expresión resguardada (como e1 → e2), la parte e1 debe ser una expresión de tipo booleano, mientras que la parte e2 puede tener cualquier tipo.

   El resultado obtenido en la parte e2 de cada expresión resguardada debe ser consistente en todas las ramas. Esto significa que si una rama produce un valor de tipo T, entonces todas las demás ramas también deben producir un valor de tipo T.

   El resultado final del operador case debe tener el mismo tipo que el resultado producido en la parte e2 de las expresiones resguardadas. En otras palabras, si todas las ramas generan valores de tipo T, el resultado del case también debe ser de tipo T.

4. Extiende el algoritmo de inferencia de tipos de la nota 8, agregando las reglas de generacion de restricciones para el operador case.

   1. Para cada expresión resguardada e1 → e2 en g:
   - Genera una restricción que requiere que e1 tenga tipo booleano.
   - Registra que e2 debe tener un tipo polimórfico, es decir, aún no se sabe su tipo específico.
   2. Para el resultado final del case (que será el tipo de la expresión completa case(t, g)):

      Registra que el tipo del resultado final debe ser el mismo que el tipo de e2 para al menos una de las ramas. Puedes hacer esto registrando una restricción de igualdad entre el tipo del resultado final y el tipo de e2 en una de las ramas.

5. Explica por que el operador case es azucar sintactica en el lenguaje.

   El operador case se considera azúcar sintáctica debido a su enfoque en mejorar la legibilidad y comodidad en la escritura de código. En muchos lenguajes, el operador case se utiliza para implementar estructuras de control de flujo condicional, como las declaraciones if o switch. Sin embargo, desde el punto de vista semántico, estas estructuras ya son suficientes para expresar condiciones y ejecutar código condicional. Por lo tanto, el operador case se puede percibir como una forma alternativa o más clara de expresar la misma lógica, simplificando la escritura del código.