Universidad Nacional Autónoma de México

Facultad de Ciencias

Lenguajes de programación

Alumnos:
Guerrero Chávez Diana Lucía
Lázaro Arias Jorge Alberto
Sánchez Alcántara Jesús Esteban

TAREA 3

Problema I

Haga el juicio de tipo para la función fibonacci y el predicado empty?

Juicio de tipo para la función fibonacci

$$\dfrac{\dfrac{\Gamma \vdash [fibo \longleftarrow number] \vdash \{fibo1\} : number}{\Gamma \vdash fibo : (number \longrightarrow number)} \quad \Gamma \vdash 1 : number \quad \dfrac{\dfrac{\dfrac{A}{\Gamma \vdash (= n0) : bool}}{\Gamma \vdash (= n0) : bool} \quad \Gamma \vdash 0 : number \quad \dfrac{B}{\Gamma \vdash *}}{\Gamma \vdash [fibo \longleftarrow number] \vdash (if(= n0)0*)}}{\Gamma \vdash [fibo \longleftarrow number] \vdash fun(n : num) : num...\}\}}$$

$$\Gamma \vdash \{rec\{fibo : number(fun(n : num) : num(if(= n0)0if(= n1)1(+(fibo(-n2))(fibo(-n1)))))\}\} : numb$$

A: $\Gamma \vdash n : number \quad \Gamma \vdash 0 : number$

B: $\dfrac{\Gamma \vdash (= n1) : bool}{\Gamma \vdash (= n1) : bool}$

* = (if(= n 1)) 1 (+(fibo(- n 2))(fibo (- n 1)))): number

...= (if (= n 0)0 (if(= n 1)) 1 (+(fibo(- n 2))(fibo (- n 1))))

$$\dfrac{\dfrac{\Gamma \vdash (-n\ 2) : number}{\Gamma \vdash (fibo(-n\ 2))} \quad \dfrac{\Gamma \vdash (-n\ 1) : number}{\Gamma \vdash (fibo(-n\ 1))}}{\Gamma \vdash\ (+(fibo(-n\ 2))(fibo(-n\ 1)))}$$

Juicio de tipo para el predicado empty?

$$\dfrac{\dfrac{\Gamma \vdash \{empty?\ l\}\ boolean}{\Gamma \vdash l : list}}{\Gamma \vdash \{empty?\ l\}\ boolean}$$

Problema II

Considera el siguiente programa:

(+ 1 (first (cons true empty)))

Este programa tiene un error de tipos.
Genera restricciones para este programa. Aísla el conjunto mas pequeño de estas restricciones tal que, resultas juntas, identifiquen el error de tipos.
Siéntete libre de etiquetar las sub-expresiones del programa con superíndices para usarlos cuando escribas y resuelvas tus restricciones.

$\boxed{1}(+\boxed{2}1\boxed{3}(\text{first}\boxed{4}(\text{cons }\boxed{5}\text{true }\boxed{6}\text{empty})))$

Restricciones:

$\boxed{2} = \boxed{3} \longrightarrow \boxed{1}$

$\boxed{3} = \boxed{4} \longrightarrow \boxed{2}$

$\boxed{4} = [|\text{cons}|] = \text{list} \longrightarrow \text{list}$

$\boxed{5} = \boxed{6} = \text{boolean}$

Sabemos que se hace una operación de suma (recibe algo de tipo number y regresa algo de tipo number).

$\boxed{3} = \boxed{4} \longrightarrow \boxed{2}$

Podemos ver que $\boxed{3}$ regresa algo de tipo boolean, por lo que la suma queda:
(+ 1 #t) y esta operación no se puede hacer porque no se puede hacer una suma de números con booleanos (number $\neq$ boolean)

Problema III

Considera la siguiente expresión con tipos:

```
{fun {f : C1 } : C2
  {fun {x : C3 } : C4
    {fun {y : C5 } : C6
      {cons x {f {f y}}}}}}
```

Dejamos los tipos sin especificar (Cn) para que sean llenados por el proceso de inferencia de tipos. Deriva restricciones de tipos para el programa anterior. Luego resuelve estas restricciones. A partir de estas soluciones, rellena los valores de las

Cn. Asegurate de mostrar todos los pasos especificados por los algoritmos (i.e., escribir la respuesta basandose en la intuición o el conocimiento es insuficiente). Deberás usar variables de tipo cuando sea necesario. Para no escribir tanto, puedes etiquetar cada expresión con una variable de tipos apropiada, y presentar el resto del algoritmo en términos solamente de estas variables de tipos.

```
{ [1] fun {f}
  { [2] fun {x}
    { [3] fun {y}
      { [4] cons [5] x  [6] {f [7] {f y}}}}}}
```

Restricciones:

$[\![\boxed{1}]\!] = [\text{f}] \rightarrow \boxed{1}$

$[\![\boxed{2}]\!] = [\text{x}] \rightarrow \boxed{2}$

$[\![\boxed{3}]\!] = [\text{y}] \rightarrow \boxed{3}$

$[\![\boxed{4}]\!] = [\{\text{cons x } \{\text{f } \{\text{f y}\}\}\}]=\text{list y } [\{\text{f } \{\text{f y}\}\}]=\text{list}$

$[\![\boxed{5}]\!] = [\text{x}] = \text{number}$

$[\![\boxed{6}]\!] = [\text{f}\{\text{f y}\}] = \text{nlist}$

$[\![\boxed{7}]\!] = [\{\text{f y}\}] = \text{nlist}$

Con las restricciones anteriores podemos inferir los tipos correspondientes a los C_n, aquí tenemos desde C_1 hasta C_6.

```
{fun {f : number } : list
  {fun {x : number } : list
    {fun {y : number } : list
      {cons x {f {f y}}}}}}
```

Problema IV

Considera los juicios de tipos discutidos en clase para un lenguaje glotón (en el capitulo de Juicios de Tipos del libro de Shriram). Considera ahora la versión perezosa del lenguaje. Pon especial atención a las reglas de tipado para:

- **definición de funciones**
- **aplicación de funciones**

Para cada una de estas, si crees que la regla original no cambia, explica por que no (Si crees que ninguna de las dos cambia, puedes responder las dos partes juntas). Si crees que algun otro juicio de tipos debe cambiar, mencionalo también.

En los lenguajes perezosos no van a cambiar los juicios de tipo, ya que el juicio de tipo que se tenga va a estar dado por los tipos(por eso es juicio de tipo) que nos vayan a dar en la función, por lo que los tipos en definición de funciones y aplicación van a ser los mismos.

Problema V

¿Cuáles son las ventajas y desventajas de tener polimorfismo explícito e implícito en los lenguajes de programación?

POLIMORFISMO EXPLÍCITO

VENTAJAS:

- Es más fácil saber a que tipo de dato se aplica la función.
- Tiene el mismo comportamiento para diferentes tipos.
- Reciclamiento de código.

DESVENTAJAS:

- Tenemos que ocupar nombres distintos para cada tipo de datos que tengamos a los cuales se les puede aplicar el método con el mismo comportamiento.
- La legibilidad que se tenga en el código no va a ser del todo clara.

POLIMORFISMO IMPLÍCITO

VENTAJAS:

- Nuestro lenguaje es más pequeño.
- No se repite código.

DESVENTAJAS:

- Es un poco más complicado tener el código.
- Tenemos que hacer verificaciones de tipos para saber como se va a comportar la función.

Problema VI

Da las ventajas y desventajas de tener lenguajes de dominio específico (DSL) y de próposito general. También da al menos tres ejemplos de lenguajes DSL, cada ejemplo debe indicar el propósito del DSL y un ejemplo documentando su uso.

Lenguajes de dominio específico(DSL)

VENTAJAS:

- A diferencia de lenguajes de propósito general, un lenguaje de dominio específico consta de los elementos y las relaciones que representan directamente la lógica del espacio del problema. Por ejemplo, una aplicación de la póliza de seguros debe incluir los elementos para las directivas y las peticiones. Un lenguaje específico facilita diseñar la aplicación, y encuentra y corregir errores de lógica.
- Mejoran la calidad, productividad, confianza, mantenibilidad y portabilidad de las aplicaciones.
- Permiten hacer validaciones a nivel del dominio. Mientras las construcciones del lenguaje estén correctas, cualquier sentencia escrita puede considerarse correcta.

DESVENTAJAS:

- Los gastos de producción pueden ser muy elevados.
- Tienen por lo general un conjunto muy restringido de características y un alto nivel de abstracción para cumplir tareas especificas.

Ejemplos:

1. SQL: tiene que ver con el acceso y uso de las bases de datos relacionalmente.

```
SELECT *
FROM Tabla;
```

2. ALGOL:es un lenguaje de programación que fue diseñado para el computo científico. Se tiene el siguiente ejemplo con el cual se puede tener la serie de fibonacci hasta un número n.

```
PROC print fibo = (INT n) VOID :
BEGIN
INT a := 0, b := 1;
FOR i TO n DO
print((whole(i, 0), -> ", whole(b, 0), new line));
INT c = a + b;
a := b;
b := c
OD
```

END;

INT i;
print("Fibonacci hasta? ");
read((i, new line));
print fibo(i)

3. CSS: es un lenguaje usado para definir y crear la presentación de un documento estructurado escrito en HTML o XML
Ejemplo:
p
text-align: center;
color: blue;

Alinea la etiqueta p y y le da un color azul.
Lenguajes de próposito general(GPL):

VENTAJAS:

- Permiten la implementación de prácticamente cualquier algoritmo.
- Pueden ser usados para varios propósitos, como el acceso a bases de datos, comunicación entre computadoras, comunicación entre dispositivos, captura de datos, cálculos matemáticos, diseño de imágenes o páginas.

DESVENTAJAS:

- Para resolver un problema en específico, como algo que tenga que ver relacionado con matemáticas es mejor usar DSL.