Lenguajes de Programación 2016-1
Tarea 3

Rodríguez Abreu José Ricardo
Martínez Martínez Julio César

23-Nov-15
Facultad de Ciencias, UNAM

Problema I

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

```
(define (fibonacci n)
        (cond
           [(equal? n 0) 0]
           [(<= n 2) 1]
           [else (+ (fibonacci (- n 1)) (fibonacci (- n 2)))]))
```

Juicio de tipo: empty:

$$\Gamma \vdash l : list$$

$$\text{----------}$$

$$\Gamma \vdash (empty?\ l) : bool$$

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 super índices para usarlos cuando escribas y resuelvas tus restricciones.

Sean:

$$^{[1]}(+\ 1\ (\text{first}\ ^{[2]}(\text{cons}\ ^{[3]}\text{true}\ ^{[4]}\text{empty})))$$

Entonces:

$\breve{}$
[[1]] ⇒ (+ 1 [2]) ⇒ (1 : number, [2] : number)
[[2]] ⇒ (cons [3] [4]) ⇒ (cons : list)
[[3]] ⇒ true : bool
[[4]] ⇒ empty : list

$\breve{}$
Pero [[2]] : number y [[2]] : list en el paso (1) y (2).

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. Asegúrate de mostrar todos los pasos especificados por los algoritmos (i.e., escribir la respuesta basándose en la intuición o el conocimiento es insuficiente). Deber´as 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.

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 1 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 algún otro juicio de tipos debe cambiar, menciónalo también.

Las principales diferencia entre un lenguaje de tipo glotón y uno de tipo perezoso son principalmente que uno evalúa expresiones cuando es posible y el otro sólo si se necesitan. Las reglas de los juicios de tipo no deberían cambiar

ya que durante el chequeo de tipos no se realiza durante el tiempo de ejecución, cuando se ejecuta el programa es cuando entran las diferencias entre el tipo de evaluación. El encargado de revisar el tipado es el intérprete o compilador, ambos funcionan de la misma manera si el lenguaje es perezoso o glotón por lo que no afecta como es la evaluación de la misma, el chequeo siempre dará lo mismo.

Problema V

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

Polimormismo explícito:

Pros:
- Evita que el código se vuelva repetitivo.
- Crea distintos objetos con las mismas características

Contras:
- Si el código requiere cambios, un pequeño cambio podría implicar un gran problema ya que muchos comportamientos podrían depender de ese cambio.
- El código es se aleja de ser legible al tener dependencias de uso (véase como consecuencia de que no se repite código).

␣

Polimorfismo implícito:

Pros:
- No hay restricciones respecto a los tipos de datos que cada objeto usa.
- No hay un estándar por lo que el código es más flexible.

Contras:
- Puede haber problemas de pérdida de información como consecuencia de la falta de estándar.

Problema VI

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

Venatajas de DSL:

- Los conocimientos adquiridos necesarios solucionan el problema que resuelve el DSL de manera más rápida y sin "rodeos".
- Al ser menos extenso que un lenguaje de propósito general, es más rápido y fácil de dominar para resolver problemas.

Desventajas de DSL:

- Algunos dependen de un lenguaje de propósito general para resolver problemas más generales.
- Falta de información y APIs.

Ventaja de lenguajes de propósito general:

- (En general) control sobre la mayoría de las funciones del objeto con el que se quiere trabajar.
- Puedes crear un DSL desde el lenguaje de propósito general y resolver el problema de la misma manera que un DSL lo haría.

Desventajas de lenguajes de propósito general:

- Sobre carga de información si no se tiene conocimientos previos.
- Al resolver problemas de dominio específico de un lenguaje DSL puede traer problemas al no ser más eficaz para resolver éstos.

Ejemplos:
Gestionado matrices para hacer transformaciones lineales en openGL

```
#version 140
uniform Transformation {
        mat4 projection_matrix;
        mat4 modelview_matrix;
};

in vec3 vertex;

void main(void) {
        gl_Position = projection_matrix * modelview_matrix * vec4(vertex, 1.0);
}
```

Código para ver el siguiente buffer disponible en emacs programado en Emacs Lisp

```
(defun my-split-window-func ()
  (interactive)
  (split-window-vertically)
  (set-window-buffer (next-window) (other-buffer)))

  (global-set-key "\C-x2" 'my-split-window-func )
```

Función creada para marcar una regla de movimiento hecha en Game Description Language (GDL).

```
(<= (legal ?player (mark ?m ?n))
    (true (cell ?m ?n blank))
    (true (control ?player)))
```