

Universidad Nacional Autónoma de México

Facultad de Ciencias

Lenguajes de Programación

Tarea 3

PROBLEMA I

FIBONACCI

$$\begin{array}{c}
 \frac{n: \text{number} \quad 2: \text{number}}{\Gamma|-n: \text{number} \quad \Gamma|-2: \text{number}} \quad \frac{\frac{n: \text{number} \quad 1: \text{number}}{\Gamma|-n: \text{number} \quad \Gamma|-1: \text{number}}}{\Gamma|-(-n \ 1): \text{number}} \quad \frac{\frac{n: \text{number} \quad 2: \text{number}}{\Gamma|-n: \text{number} \quad \Gamma|-2: \text{number}}}{\Gamma|-(-n \ 2): \text{number}} \\
 \frac{\Gamma(\text{fib}[number \rightarrow number])|-(-n \ 1): \text{number}}{\Gamma|-(\text{fib}(-n \ 1)): \text{number}} \quad \frac{\Gamma(\text{fib}[number \rightarrow number])|-(-n \ 2): \text{number}}{\Gamma|-(\text{fib}(-n \ 2)): \text{number}} \\
 \frac{\Gamma|-(\text{fib}(-n \ 1)): \text{number} \quad \Gamma|-(\text{fib}(-n \ 2)): \text{number}}{\Gamma|-(+ (\text{fib}(-n \ 1)) (\text{fib}(-n \ 2))): \text{number}} \\
 \frac{\Gamma|-(\leq n \ 2): \text{bool} \quad \Gamma[\text{fib}(number)]|-(\text{cond}[(\leq n \ 2)][\text{else} (+ (\text{fib}(-n \ 1)) (\text{fib}(-n \ 2)))]): \text{number} \rightarrow \text{number}}{\Gamma|-(\text{rec}(f \text{ ivo}: \text{number}(\text{cond}[(\leq n \ 2)1][\text{else} (+ (\text{fib}(-n \ 1)) (\text{fib}(-n \ 2)))]))}
 \end{array}$$

$$\begin{array}{c}
 \text{EMPTY} \\
 \frac{\Gamma|-l: \text{List}}{\Gamma|-(\text{Empty} ? l): \text{Bool}}
 \end{array}$$

PROBLEMA II

Tenemos la función `suma (+ exp1 exp2)` donde `exp1` y `exp2` son de tipo `number` entonces con `(first (cons true empty))` debemos devolver `number` y con `(cons true empty)` devolvemos un `nlist`.

```

[[1]] (+ [[2]] 1 [[3]] (first [[4]] (cons [[5]] true [[6]] empty)))
[[1]] = [(+ [[2]] 1 [[3]] (first [[4]] (cons [[5]] true [[6]] empty)))]→number
[[2]] = [1]→number
[[3]] = [(first [[4]] (cons [[5]] true [[6]] empty))] → number
[[4]] = [(cons [[5]] true [[6]] empty)] → nlist
[[5]] = [true]→bool
[[6]] = [empty]→nlist

```

En `[[5]]` vemos que se está intentando construir una lista de números con un tipo `bool` lo cual causaría un error

PROBLEMA III

```

[[1]] = [F] → number
[[2]] = [x] → number
[[3]] = [y] → number
[[4]] = [cons x {f {f y}}] → list ya que [x]→number y [{f {f y}}]→list
[[5]] = [x]→number

```

$\llbracket 6 \rrbracket = [\{f \{f y\}\}] \rightarrow \text{list}$
 $\llbracket 7 \rrbracket = [\{f y\}] \rightarrow \text{nlist}$

Por lo tanto C_1 , C_3 , C_5 son tipo number y C_2 , C_4 , C_6 son nlist.

PROBLEMA IV

No importa si es perezoso o glotón Los juicios de tipo no cambian ya que el chequeo de tipos no se realiza en tiempo de ejecución, lo hace el compilador o interprete, entonces la evaluación no afecta el chequeo de tipos.

PROBLEMA V

Polimorfismo Explícito

Ventajas:

- Es más rápido en tiempo de compilación.
- Podemos usar el mismo código para varios propósitos.

Desventajas

- Escribir el código fuente es más tardado.
- una variable no puede ser utilizada para distintos tipos lo que significar más variables y por ende más espacio en memoria ocupado.

Polimorfismo Implícito

Ventajas

- No hay necesidad de construir un tipo en específico ya que siempre que usemos variables estaremos trabajando con tipos genéricos.
- El código fuente es mucho menos verboso.

Desventajas

- Los errores de tipo se dan en tiempo de ejecución.
- El programa es más propenso a errores de semántica.

PROBLEMA VI

DSL

Ventajas

- Fácil para resolver problemas específicos ya que es más "liviano" que un lenguaje de propósito general.
- Resuelven el problema sin redundancias.

Desventajas:

- Los problemas más grandes requieren el uso de un lenguaje de propósito general

Lenguajes de propósito general

Ventajas

- DSL dentro del mismo lenguaje de propósito general.
- Mayor control sobre el programa.

Desventajas:

- A veces hay muchas cosas innecesarias.
- A veces es menos eficaz para resolver problemas pequeños y específicos.