Tarea Independiente 09/10/2025

David Núñez Franco

October 12, 2025

Inventario de Conceptos Claves

- Visitador de AST de código que genera un AST de tipos
- Tipo como AST
- Compatibilidad de tipos
- Scope como ambiente (environment, o Env).
- Modelo de JS como máquina asincrónica (monohilo, pila cola)
- Loop de eventos (Event-loop)
- Timers

Ejercicio 0

Estudie el mini-typer entregado y añada el caso de TupleType.

Solución

```
        // types as AST
        export class Type extends Node {
            toString() {
                return this.head.toString();
            }
        }

        export class IntType extends Type {
            constructor() {
                super("int");
            }
        }

```

```
14          export class AnyType extends Type {
15              constructor() {
16                  super("any");
17              }
18          }
19
20          export class TupleType extends Type {
21              constructor(...elemTypes) {
22                  super("tuple", ...elemTypes);
23              }
24              get elems() {
25                  return this.children;
26              }
27              toString() {
28                  return `(${this.elems.map((t) => t.toString()).join(
                        ", ")})`;
29              }
30          }
31
32          export class AstPrintVisitor extends Visitor {
33              visit(node) {
34                  if (!node) {
35                      return;
36                  }
37                  if (node instanceof Num) {
38                      console.log(node.value);
39                      return;
40                  }
41                  if (node instanceof Id) {
42                      console.log(node.name);
43                      return;
44                  }
45
46                  if (node instanceof Type) {
47                      console.log(node.toString());
48                      return;
49                  }
50
51                  if (node instanceof UnaryOp) {
52                      console.log(node.oper.name);
53                      console.log(node.expr.accept(this));
54                      return;
55                  }
56              }
57          }
```

Listing 1: ast.mjs

```
        function test_case_tuple_type() {
            const tInt = new IntType();
            const tAny = new AnyType();
            const tPair = new TupleType(tInt, tInt);
            const tTriple = new TupleType(tInt, tAny, tPair);

            console.log("TupleType pair =", tPair.toString());
            console.log("TupleType triple =", tTriple.toString());

            const printer = new AstPrintVisitor();
            tTriple.accept(printer);
        }
```

Listing 2: main.mjs

```
        TupleType pair = (int, int)
        TupleType triple = (int, any, (int, int))
        (int, any, (int, int))
```

Listing 3: output

Ejercicio 2

Considere este código
function print_json(user=1){
 fetch(‘https://jsonplaceholder.typicode.com/todos/${user}‘)
 .then(response => response.json())
 .then(json => console.log(json))
 .catch(err => console.error(err))
}
Reescriba usando código asincrónico imperativo async/await de manera equivalente.

Solución

```
        async function print_json(user = 1) {
            try {
                const response = await fetch(
                ‘https://jsonplaceholder.typicode.com/todos/${user}‘
                );
                const json = await response.json();
                console.log(json);
            } catch (err) {
                console.error(err);
            }
```

```
11          }
```

Listing 4: async/wait code

Ejercicio 3

```
Escriba en JS una funcción asincrónica repeat(action, conditon, secs) que cumpla lo si
action y condition son lambdas asincrónicas tales que:
action() devuelve una Promise que resuelve con un  value.
Entonces condition(value) devuelve una Promise que resuelve con true o false.
La función repeat debe esperar al menos secs segundos antes de comenzar.
Ejecuta repetidamente action(). Se detiene cuando el resultado devuelto por action cum
Al terminar repeat retornará el primer value devuelto por action que cumplió condition
Use async/await y por ende (para variar) código imperativo en este caso.
Ejemplo de uso: genere el primer random mayor que .9,
repeat(
async () => Math.random(),
async x => x > 0.9,
2                    // espere 2 segundos antes de emepzar a generar
).then(x => console.log(‘First value found ${x.toFixed(3)}‘))
```

Solución

```
1           async function repeat(action, condition, secs) {
2               // esperar al menos secs segundos antes de iniciar
3               await new Promise((resolve) => setTimeout(resolve, secs
                    * 1000)); // crea la espera inicial
4
5               // bucle controlado por la condicion
6               // cada iteracion espera la resolucion de action(), y
                    luego evalua condition(value)
7               // cuando condition devuelve true, se retorna ese valor
                    y la funcion termina
8               while (true) {
9                   const value = await action();
10                  const ok = await condition(value);
11                  if (ok) return value;
12              }
13          }
14
15          repeat(
16          async () => Math.random(),
17          async (x) => x > 0.9,
18          2
```

```
19          ).then((x) => console.log(`First value found ${x.toFixed(3)
               }`));
```