Raúl Rubio Cuerpo, Adrián Contreras Martín

INFORME PRÀCTICA CAP 23/24 Q1

1.Introducció:

Com aquesta pràctica anava de continuacions amb rhino, vam començar repassant la teoria sobre aquest tema per tenir tots els conceptes clar. Al llegir l'enunciat,vam veure que s'havia d'implementar el catch/throw de Common Lisp, i vam posar-nos a buscar informació sobre aquest per saber-ne ben bé que havíem de fer.

2.Explicació del codi:

Farem una explicació de les parts del codi, que ho dividirem en 3 parts, els atributs més funció auxiliar, codi del my_catch y per últim el codi del my_throw.

- **Atributs i funció auxiliar:**

```
let { my_catch, my_throw } =

  ( function () {

      // ... les dades/atributs/estat necessaris (a decidir per vosaltres)
      let pila = []

      // ... les funcions auxiliars que us facin falta (a decidir per vosaltres)
      function findElement(a,t) {
        for (let i = 0; i < a.length; ++i) {
          let tag = a[i].tag
          if (tag === t) return i
        }
        return -1
      }
```

Com podem veure, tenim un atribut que es un array on cada posició d'aquest és un objecte que conté un parell que és la continuació i la seva tag. Després, tenim una funció auxiliar per buscar la tag dins d'aquest atribut, ja que la funció que porta JavaScript no acabava de funcionar-nos bé.

- **My_catch:**

```
function _catch(tag, fun) {
  const found = findElement(pila,tag)
  if (found === -1) pila.push({'tag': tag, 'cont': current_continuation()})
  else throw("No es permeten tags repetits")
  if (pila[pila.length-1].cont instanceof Continuation) {
    let val = fun()
    pila.pop()
    return val

  }
  else {
    return pila.pop().cont
  }

}
```

Aquest és el nostre codi del my_catch. El seu funcionament és el següent:

1. Primerament, comprovarem que el tag que ens proporciona el my_catch no sigui repetit, en cas que sigui, llancem el throw, en cas que no, s'afegeix a la pila amb la seva continuació corresponent. D'aquesta manera complim amb la condició de l'enunciat que diu que no es permeten tags repetits.
2. Després, tenim un if per comprovar si l'últim element de la pila, al seu parell {tag,cont} si cont es una continuació (el primer cop sempre entra) per poder executar la funció que es pasa com a paràmetre al my_catch.
3. Dins de l'if, depenent de la funció fun, si té un my_throw o es una operació simple (+,*,-,...) tindrà diferents comportaments:
    - Si es una operació normal, simplement retorna el valor, s'elimina l'últim element de la pila i es retorna el valor.
    - En cas que fun sigui un my_throw, seguirá el comportament que explicarem més avall del my_throw.
4. En el else, trobem el cas que en el parell {tag,cont} cont ja no és una continuació(això canvia dins del my_throw que explicarem més avall) i llavors simplement retornem el valor eliminant-lo de la pila pel qualsevol problema a posterior(amb els diferents tests).

- **My_throw:**

```
    function _throw(tag, val) {
        // Pre: tag és una String; val és qualsevol valor (però no una Continuation!)
        // ...
        const index = findElement(pila,tag)
        if (index !== -1) {
          let c = pila[index].cont
          pila.splice(index, pila.length)
          c(val)
        }
        else {
          throw("my_throw no tiene my_catch")
        }

    }

    return { my_catch: _catch, my_throw: _throw }

}())
```

El primer que fem és utilitzar la funció auxiliar que vam definir per saber-ne si la tag está dins de la nostra pila(-1 si no es troba, qualsevol altre valor es la posició dins de la pila).

Després comprovem si está o no, on hi ha diferents comportaments:

- En cas de que si es trobi(index !== -1) agafem la continuació corresponent al tag, invalidem tots els elements de la pila posteriors al tag que hem trobat,d'aquesta manera complim la condició de l'enunciat "*tots els tags associats a my_catch posteriors al my_catch on estem retornant han de quedar invalidats*". Després, cridem a la continuació amb el valor que es pasa a my_throw(així la continuació passa a ser un valor i així fem que entri al else del my_catch).
- En cas contrari, llencem una excepció per complir amb l'enunciat: *"També podem fer my_throw a tags que ja no existeixen (perquè ja hem acabat l'execució del thunk associat al my_catch que crea el tag) o que no s'han creat mai amb cap crida a my_catch. En aquest cas cal generar un error."*

3.Tests:

Principalment, hem fet servir els tests proporcionats a l'enunciat de la pràctica. Vam anar progressivament testejant els tests individualment i a mesura que anaven funcionant feiem com uns tests integrant més d'un fins que hem pogut executar tots amb els outputs corresponents. A més, vam anar fent molts prints per veure l'estat de la pila en parts claus del codi per veure que funcionava com desitjavem.
Després, vam fer un tests propis per assegurar-nos del correcte funcionament del codi:

```
// d
try {
  my_throw('etiqueta', 100);
} catch (error) {
  print(error); // Debería imprimir "my_throw no tiene my_catch"
}

// e -> Hauría d'imprimir 100
print(my_catch('zero', function () {
  return my_catch('one', function () {
    return my_catch('two', function () {
      return my_throw('one', 100);
    });
  });
}));


// f -> Hauría d'imprimir l'error del throw(tags repetits)
try {
  print(my_catch('zero', function () {
    return my_catch('zero', function () {
      return my_catch('two', function () {
        return my_throw('one', 100);
      });
    });
  }));
}catch(error) {
  print(error)
}
```

Al test d) el que volem comprovar es que si es llença un my_throw sense un previ my_catch, ha de llençar un error de que aquest my_throw no té my_catch.

El test e) volem comprovar que l'anidament de my_catch sense tags repetits funciona correctament.
I per últim, el test f) volem comprovar que l'anidament de my_catch amb tags repetits no funciona i ha de llençar l'error de que no es permeten tags repetits.

4.Dificultats:

Primerament, la primera dificultat va ser que capturàvem només una continuació per execució en comptes de una per cada my_catch, llavors això feia que els primers testos on només es feia un sol my_catch funcionés, però al test c on hi havien més d'un my_catch, no funcionés bé.
Després, vam tenir problemes amb el test que fa print(test(0)), ja que no teniem implementat el throw per llençar l'error, teniem un simple 'return' que això feia que ens retornés un NaN que al seu moment va ser un mal de cap.
També, vam tenir un problema quan executàvem els primers dos testos de seguit, ja que havia d'imprimir "302 100" i ens imprimia "302 100 100", i no sabíem el perquè, fins que ens vam adonar que era perquè quan fèiem el my_catch que retornava 302, no eliminàvem bé les tags i es quedava penjada una altre tag que ens feia retornar el 100 extra. Ara ens hem adonat que això funcionava perquè no comprovàvem si havia tags repetits, que igualment faltava eliminar bé els tags però si haguéssim tingut aquesta comprovació podríem haver resolt aquest problema més ràpid.

5.Què hem après ?

Per concluir, podem dir que hem aprés el següent:

1. Ara, prenem més consciencia a l'hora de debugar el codi, ja que al controlar manualment un mateix el fluxe del codi, s'ha de tenir més cura.
2. Hem entès millor com funcionen les continuacions, ja que em passat de la teoria a la pràctica tot el que em après.
3. Ens hem adonat compte de lo potent que son les continuacions, ja que es pot implementar qualsevol estructura de control i et dóna el control total de com s'executa el codi.
4. I per últim, hem après com funciona el catch/throw de Common Lisp.

6. Webgrafia:

«Error Handling Common Lisp has the throw/catch statements so that you can simulate exception handling as seen in languages like Java But CL goes far beyond. - ppt download». Accedido 29 de diciembre de 2023. https://slideplayer.com/slide/7346708/.