

# Estructuras de datos

Clases teóricas

por Pablo E. “Fidel” Martínez López

## 3. Tipos algebraicos recursivos

**Repaso**

# Tipos algebraicos

- ❑ Los tipos algebraicos son tipos nuevos
  - ❑ Se definen a través de ***constructores***
    - ❑ Los constructores pueden llevar argumentos
    - ❑ Cada constructor expresa a un grupo de elementos
  - ❑ Se acceden mediante ***pattern matching***
    - ❑ Los constructores se usan para preguntar

# Tipos algebraicos

- ❑ Los tipos algebraicos se clasifican en
  - ❑ enumerativos
    - ❑ varios constructores sin argumentos (e.g. **Direccion**)
  - ❑ registros o productos
    - ❑ un único constructor con varios argumentos (e.g. **Persona**)
  - ❑ sumas o variantes
    - ❑ varios constructores con argumentos (e.g. **Helado**)
  - ❑ recursivos
    - ❑ suma que usa el mismo tipo como argumento (e.g. **Listas**)

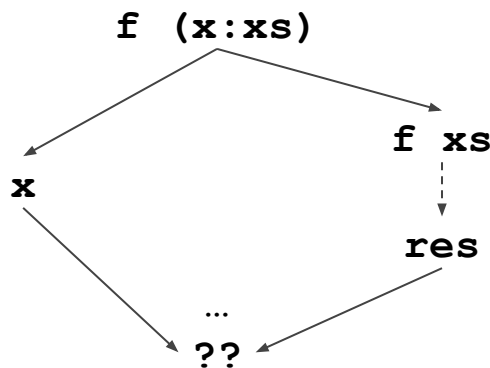
# Listas

- Las listas son un tipo algebraico predefinido
  - Tiene sintaxis especial
  - Los constructores son
    - `[] :: [a]` `-- Nil`
    - `(:) :: a -> [a] -> [a]` `-- Cons`
  - Se usa notación especial para simplificar
    - `[10,20,30]` es en realidad `(10:20:30:[])`
  - Para definir funciones hace falta recursión estructural

# Recursión estructural sobre listas

- Las definiciones recursivas estructurales sobre listas
  - Tienen un caso por cada constructor
  - En el recursivo, usan *la misma función* en la parte recursiva
  - Solo hace falta pensar cómo agregar lo que falta

```
f :: [a] -> b  
f []      = ...  
f (x:xs) = ... x ... f xs ...
```



# **Tipos algebraicos recursivos lineales**

# Otros tipos recursivos lineales

- ¿Cómo se definen tipos algebraicos recursivos?
  - Igual que otros tipos algebraicos
  - Pero se utiliza como argumento el tipo que se define
    - Solamente en *algunos* de los constructores
    - Los otros constituyen casos base

```
data Ingrediente = Salsa | Queso | Aceitunas Int  
                 | Anchoas | Anana | Roquefort | ...
```

```
data Pizza = Prepizza | Capa Ingrediente Pizza
```

- ¿Qué forma tienen los elementos de **Pizza**?



# Otros tipos recursivos li

```
data Ingrediente = Salsa | Queso
                  | Aceitunas Int
                  | Anchoas | Anana | ...

data Pizza = Prepizza
            | Capa Ingrediente Pizza
```

❏ ¿Qué forma tienen los elementos de **Pizza**?

❏ Empiezan con un constructor

❏ Tienen argumentos que ya existan

```
pizza0 = Prepizza
```

```
pizza1 = Capa Salsa Prepizza
```

```
pizza2 = Capa Queso (Capa Salsa Prepizza)
```

```
pizza3 = Capa (Aceitunas 8)
          (Capa Queso (Capa Salsa Prepizza))
```

```
pizza4 = Capa Anana (Capa Queso Prepizza)
```

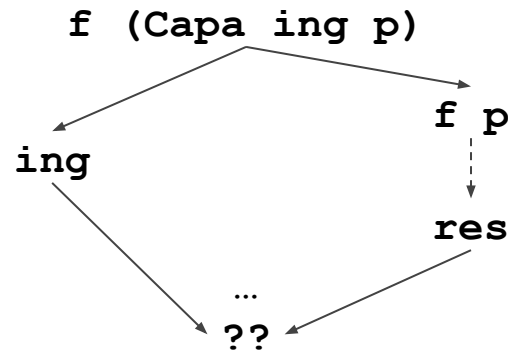
# Otros tipos recursivos li

```
data Ingrediente = Salsa | Queso
                  | Aceitunas Int
                  | Anchoas | Anana | ...

data Pizza = Prepizza
           | Capa Ingrediente Pizza
```

- ¿Y las funciones sobre **Pizzas**?
- Usamos *recursión estructural*
  - Un caso por cada constructor, y se puede usar la misma función que se define sobre la partes recursivas

```
f :: Pizza -> a
f Prepizza      = ...
f (Capa ing p) = ... ing ... f p ...
```



# Otros tipos recursivos li

```
data Ingrediente = Salsa | Queso
                  | Aceitunas Int
                  | Anchoas | Anana | ...

data Pizza = Prepizza
           | Capa Ingrediente Pizza
```

■ ¿Y las funciones sobre **Pizzas**?

■ Usamos recursión estructural

```
cantQueso :: Pizza -> Int
cantQueso Prepizza      = ...
cantQueso (Capa ing p) = ... ing ... cantQueso p ...
```

# Otros tipos recursivos li

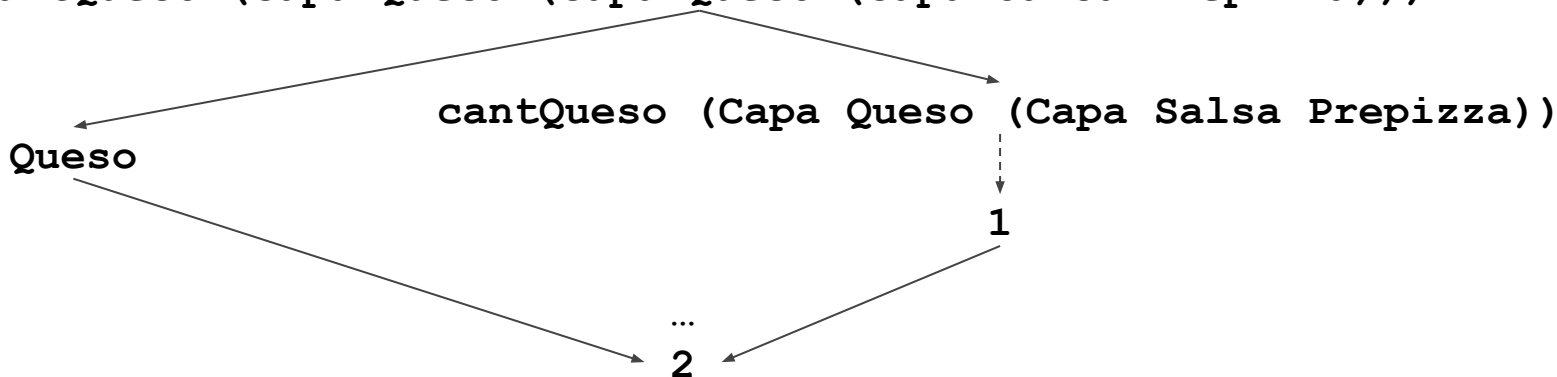
```
data Ingrediente = Salsa | Queso
                  | Aceitunas Int
                  | Anchoas | Anana | ...

data Pizza = Prepizza
            | Capa Ingrediente Pizza
```

■ ¿Y las funciones sobre **Pizzas**?

■ Usamos recursión estructural

```
cantQueso (Capa Queso (Capa Queso (Capa Salsa Prepizza)))
```



# Otros tipos recursivos li

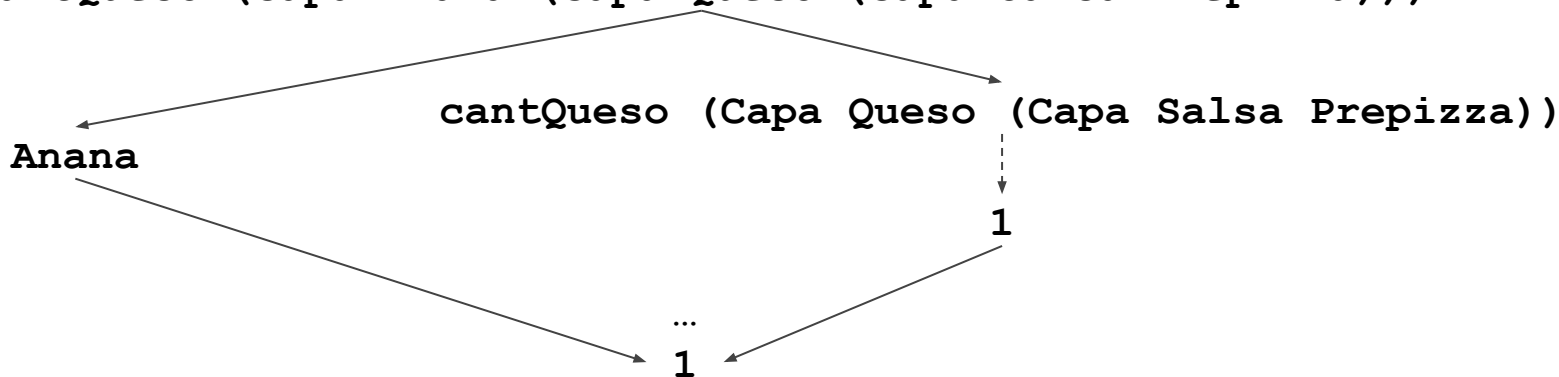
```
data Ingrediente = Salsa | Queso
                  | Aceitunas Int
                  | Anchoas | Anana | ...

data Pizza = Prepizza
            | Capa Ingrediente Pizza
```

■ ¿Y las funciones sobre **Pizzas**?

■ Usamos recursión estructural

```
cantQueso (Capa Anana (Capa Queso (Capa Salsa Prepizza)))
```



# Otros tipos recursivos li

```
data Ingrediente = Salsa | Queso
                  | Aceitunas Int
                  | Anchoas | Anana | ...

data Pizza = Prepizza
           | Capa Ingrediente Pizza
```

- ¿Y las funciones sobre **Pizzas**?

- Usamos recursión estructural

```
cantQueso :: Pizza -> Int
cantQueso Prepizza      = ...
cantQueso (Capa ing p) = unoSiQueso ing + cantQueso p

unoSiQueso :: Ingrediente -> Int
...
```

- Primero se resuelve el caso recursivo

- ¡Se pueden utilizar funciones auxiliares!

# Otros tipos recursivos li

```
data Ingrediente = Salsa | Queso
                  | Aceitunas Int
                  | Anchoas | Anana | ...

data Pizza = Prepizza
           | Capa Ingrediente Pizza
```

■ ¿Y las funciones sobre **Pizzas**?

■ Usamos recursión estructural

```
cantQueso :: Pizza -> Int
cantQueso Prepizza      = ...
cantQueso (Capa ing p) = unoSiQueso ing + cantQueso p

unoSiQueso :: Ingrediente -> Int
unoSiQueso Queso = ...
unoSiQueso _     = ...
```

■ Primero se resuelve el caso recursivo

■ ¡Se pueden utilizar funciones auxiliares!

# Otros tipos recursivos li

```
data Ingrediente = Salsa | Queso
                  | Aceitunas Int
                  | Anchoas | Anana | ...

data Pizza = Prepizza
           | Capa Ingrediente Pizza
```

- ¿Y las funciones sobre **Pizzas**?

- Usamos recursión estructural

```
cantQueso :: Pizza -> Int
cantQueso Prepizza      = ...
cantQueso (Capa ing p) = unoSiQueso ing + cantQueso p

unoSiQueso :: Ingrediente -> Int
unoSiQueso Queso = 1
unoSiQueso _     = 0
```

- Primero se resuelve el caso recursivo

- ¡Se pueden utilizar funciones auxiliares!



## Otros tipos recursivos li

```
data Ingrediente = Salsa | Queso
                  | Aceitunas Int
                  | Anchoas | Anana | ...

data Pizza = Prepizza
            | Capa Ingrediente Pizza
```

■ ¿Y las funciones sobre **Pizzas**?

■ Usamos recursión estructural

```
cantQueso :: Pizza -> Int
cantQueso Prepizza      = 0
cantQueso (Capa ing p) = unoSiQueso ing + cantQueso p

unoSiQueso :: Ingrediente -> Int
unoSiQueso Queso = 1
unoSiQueso _     = 0
```

■ Finalmente se resuelve el caso base

# Otros tipos recursivos li

```
data Ingrediente = Salsa | Queso
                  | Aceitunas Int
                  | Anchoas | Anana | ...

data Pizza = Prepizza
           | Capa Ingrediente Pizza
```

## ■ Más funciones sobre **Pizzas**

```
cantAceitunas :: Pizza -> Int
  -- cantAceitunas (Capa (Aceitunas 4) (Capa Anchoas
  --               (Capa (Aceitunas 3) (Capa Queso Prepizza))) = 7

agregados :: Pizza -> [Ingrediente]
  -- agregados (Capa Queso (Capa Anchoas (Capa Queso
  --               (Capa (Aceitunas 2) (Capa Salsa Prepizza)
  --               = [ Anchoas, Aceitunas 2 ]


duplicarQueso :: Pizza -> Pizza
  -- duplicarQueso (Capa Anana (Capa Queso Prepizza))
  --       = Capa Anana (Capa Queso (Capa Queso Prepizza))
```

**Árboles**

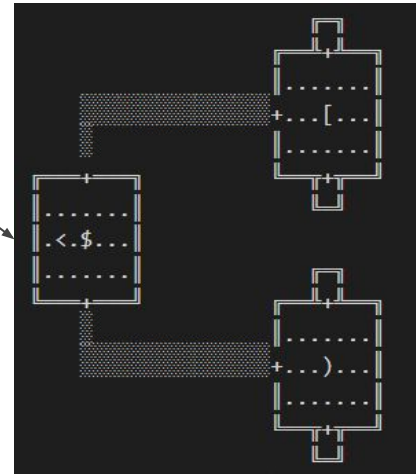
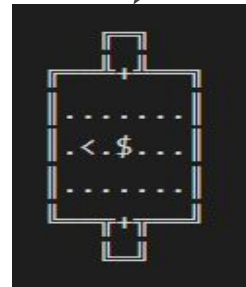
# Árboles

■ ¿Cómo serían representaciones más complejas?

■ **Ejemplo:** representar un *dungeon* (calabozo)

- Puede ser un armario vacío —————→ 
- Puede ser una habitación con 2 puertas que llevan a otras partes del *dungeon* y conteniendo un objeto

OBJETOS  
[ - Armadura  
) - Escudo  
% - Maza  
\$ - Oro



# Árboles

- ¿Cómo serían representaciones más complejas?
- Exactamente igual: constructores con argumentos

```
data Objeto = Armadura | Escudo | Maza | Oro
```

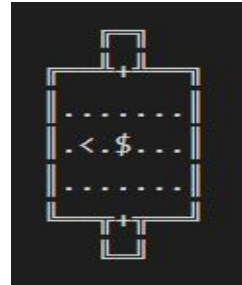
```
data Dungeon = Armario  
              | Habitacion Objeto Dungeon Dungeon
```

OBJETOS  
[ - Armadura  
) - Escudo  
% - Maza  
\$ - Oro

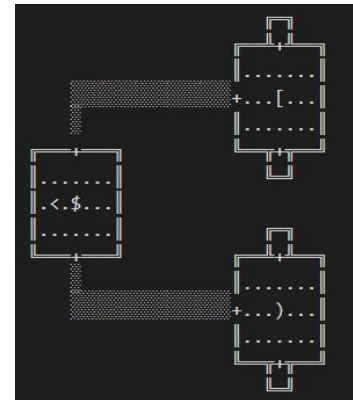
d1



d2



d3



# Árboles

```
data Objeto = Armadura | Escudo  
            | Maza      | Oro  
  
data Dungeon = Armario  
              | Habitacion Objeto  
              Dungeon Dungeon
```

- ¿Cómo serían representaciones más complejas?
- Exactamente igual: constructores con argumentos

d1 = ??

d2 = ??

d3 = ??

d1



# Árboles

```
data Objeto = Armadura | Escudo  
            | Maza      | Oro  
  
data Dungeon = Armario  
              | Habitacion Objeto  
              Dungeon Dungeon
```

- ¿Cómo serían representaciones más complejas?
- Exactamente igual: constructores con argumentos

```
d1 = Armario
```

```
d2 = ??
```

```
d3 = ??
```

d1



# Árboles

```
OBJETOS  
[ - Armadura  
) - Escudo  
% - Maza  
$ - Oro
```

```
data Objeto = Armadura | Escudo  
            | Maza      | Oro  
  
data Dungeon = Armario  
              | Habitacion Objeto  
              Dungeon Dungeon
```

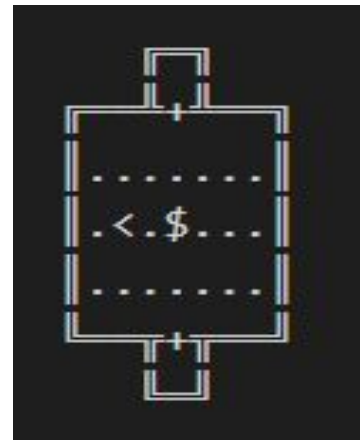
- ¿Cómo serían representaciones más complejas?
- Exactamente igual: constructores con argumentos

```
d1 = Armario
```

```
d2 = ??
```

```
d3 = ??
```

d2





# Árboles

```
OBJETOS  
[ - Armadura  
) - Escudo  
% - Maza  
$ - Oro
```

```
data Objeto = Armadura | Escudo  
            | Maza      | Oro  
  
data Dungeon = Armario  
              | Habitacion Objeto  
              Dungeon Dungeon
```

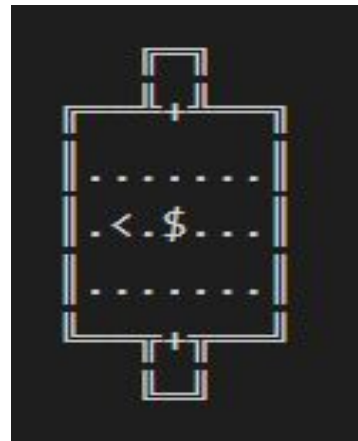
- ¿Cómo serían representaciones más complejas?
- Exactamente igual: constructores con argumentos

```
d1 = Armario
```

```
d2 = Habitacion ?? ?? ??
```

```
d3 = ??
```

d2



# Árboles

```
OBJETOS  
[ - Armadura  
) - Escudo  
% - Maza  
$ - Oro
```

```
data Objeto = Armadura | Escudo  
            | Maza      | Oro  
  
data Dungeon = Armario  
              | Habitacion Objeto  
              Dungeon Dungeon
```

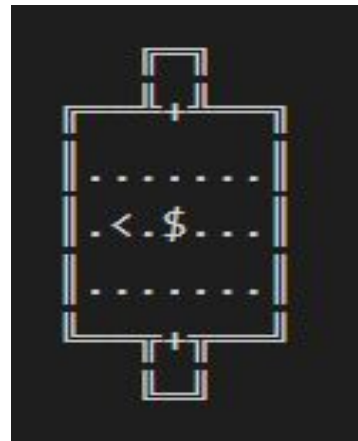
- ¿Cómo serían representaciones más complejas?
- Exactamente igual: constructores con argumentos

```
d1 = Armario
```

```
d2 = Habitacion Oro ?? ??
```

```
d3 = ??
```

d2



# Árboles

```
OBJETOS  
[ - Armadura  
) - Escudo  
% - Maza  
$ - Oro
```

```
data Objeto = Armadura | Escudo  
            | Maza      | Oro  
  
data Dungeon = Armario  
              | Habitacion Objeto  
              Dungeon Dungeon
```

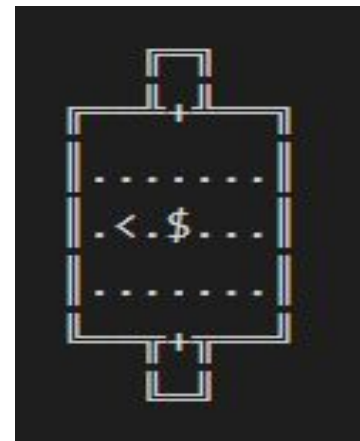
- ¿Cómo serían representaciones más complejas?
- Exactamente igual: constructores con argumentos

```
d1 = Armario
```

```
d2 = Habitacion Oro Armario Armario
```

```
d3 = ??
```

d2



# Árboles

```
OBJETOS  
[ - Armadura  
) - Escudo  
% - Maza  
$ - Oro
```

```
data Objeto = Armadura | Escudo  
            | Maza      | Oro  
  
data Dungeon = Armario  
              | Habitacion Objeto  
              Dungeon Dungeon
```

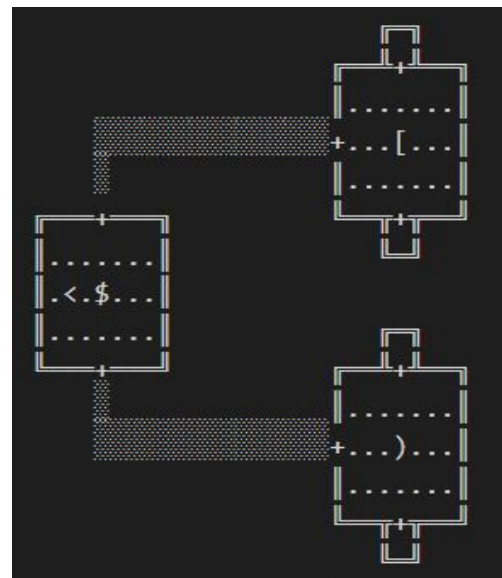
- ¿Cómo serían representaciones más complejas?
- Exactamente igual: constructores con argumentos

```
d1 = Armario
```

```
d2 = Habitacion Oro Armario Armario
```

```
d3 = ??
```

d3



# Árboles

```
OBJETOS  
[ - Armadura  
) - Escudo  
% - Maza  
$ - Oro
```

```
data Objeto = Armadura | Escudo  
            | Maza      | Oro  
  
data Dungeon = Armario  
              | Habitacion Objeto  
              Dungeon Dungeon
```

❏ ¿Cómo serían representaciones más complejas?

❏ Exactamente igual: constructores con argumentos

```
d1 = Armario
```

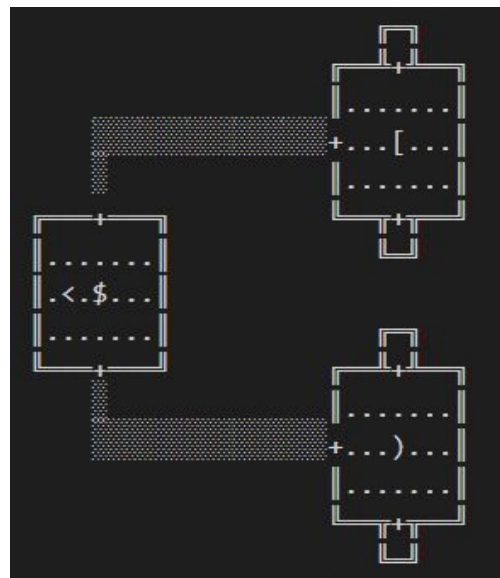
```
d2 = Habitacion Oro Armario Armario
```

```
d3 = Habitacion Oro
```

```
??
```

```
??
```

d3



# Árboles

```
OBJETOS  
[ - Armadura  
) - Escudo  
% - Maza  
$ - Oro
```

```
data Objeto = Armadura | Escudo  
            | Maza      | Oro  
  
data Dungeon = Armario  
              | Habitacion Objeto  
              Dungeon Dungeon
```

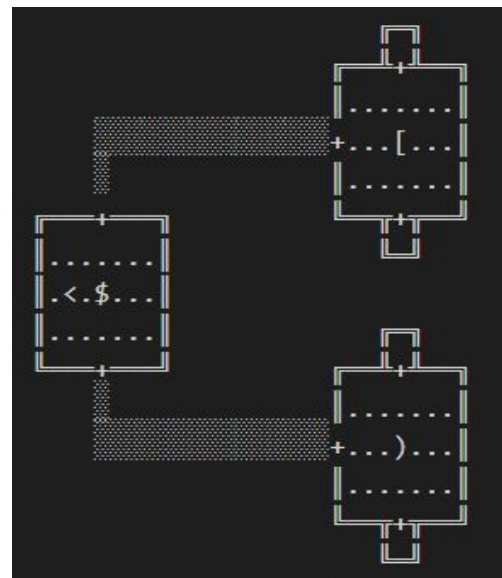
- ¿Cómo serían representaciones más complejas?
- Exactamente igual: constructores con argumentos

```
d1 = Armario
```

```
d2 = Habitacion Oro Armario Armario
```

```
d3 = Habitacion Oro  
    (Habitacion Armadura Armario Armario)  
    (Habitacion Escudo   Armario Armario)
```

d3



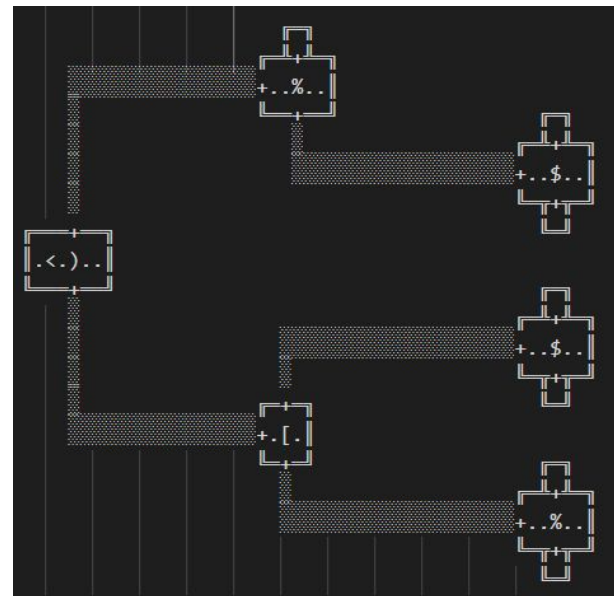
# Árboles

```
OBJETOS  
[ - Armadura  
) - Escudo  
% - Maza  
$ - Oro
```

```
data Objeto = Armadura | Escudo  
            | Maza      | Oro  
  
data Dungeon = Armario  
              | Habitacion Objeto  
              Dungeon Dungeon
```

- ¿Cómo serían representaciones más complejas?
- Exactamente igual: constructores con argumentos

```
d4 =  
  Habitacion Escudo  
    (Habitacion Maza  
      Armario  
        (Habitacion Oro  Armario Armario)  
      )  
    (Habitacion Armadura  
      (Habitacion Oro  Armario Armario)  
      (Habitacion Maza Armario Armario)  
    )
```



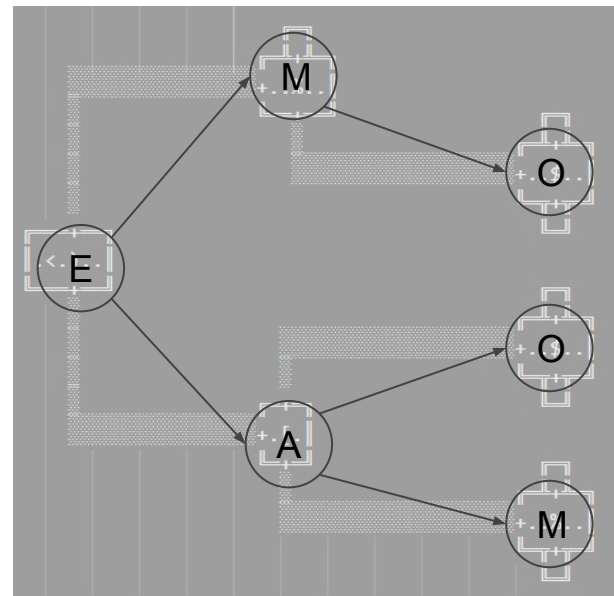
# Árboles

```
data Objeto = Armadura | Escudo
            | Maza      | Oro

data Dungeon = Armario
              | Habitacion Objeto
              Dungeon Dungeon
```

- Tipos algebraicos recursivos: árboles
- Los gráficos se suelen usar para facilitar la lectura

```
d4 = Habitacion Escudo
    (Habitacion Maza
     Armario
     (Habitacion Oro  Armario Armario)
    )
    (Habitacion Armadura
     (Habitacion Oro  Armario Armario)
     (Habitacion Maza Armario Armario)
    )
```





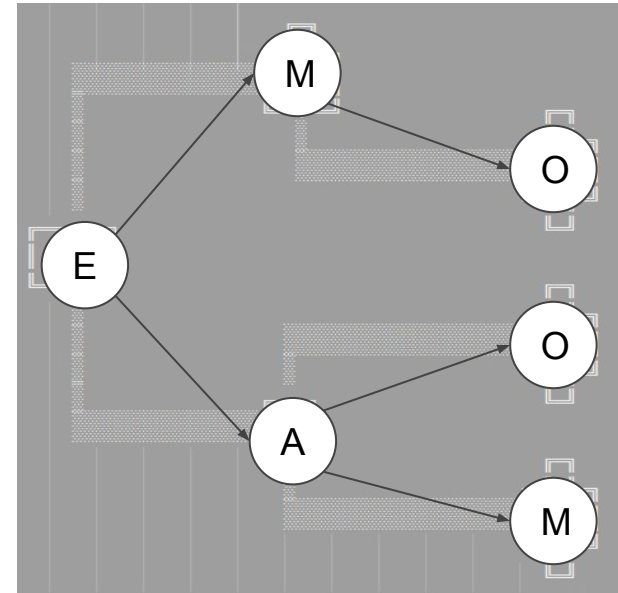
# Árboles

```
data Objeto = Armadura | Escudo
            | Maza      | Oro

data Dungeon = Armario
              | Habitacion Objeto
              | Dungeon Dungeon
```

- Tipos algebraicos recursivos: árboles
- Los gráficos se suelen usar para facilitar la lectura

```
d4 = Habitacion Escudo
    (Habitacion Maza
     Armario
     (Habitacion Oro  Armario Armario)
    )
    (Habitacion Armadura
     (Habitacion Oro  Armario Armario)
     (Habitacion Maza Armario Armario)
    )
```



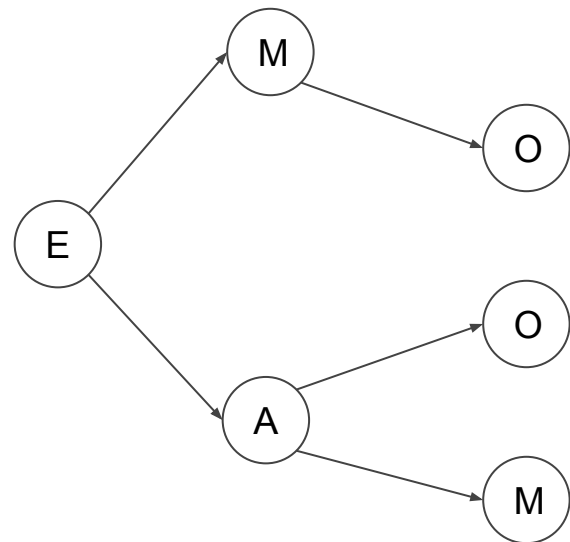
# Árboles

```
data Objeto = Armadura | Escudo
            | Maza      | Oro

data Dungeon = Armario
            | Habitacion Objeto
            | Dungeon Dungeon
```

- Tipos algebraicos recursivos: árboles
- Los gráficos se suelen usar para facilitar la lectura

```
d4 = Habitacion Escudo
    (Habitacion Maza
     Armario
     (Habitacion Oro  Armario Armario)
    )
    (Habitacion Armadura
     (Habitacion Oro  Armario Armario)
     (Habitacion Maza Armario Armario)
    )
```



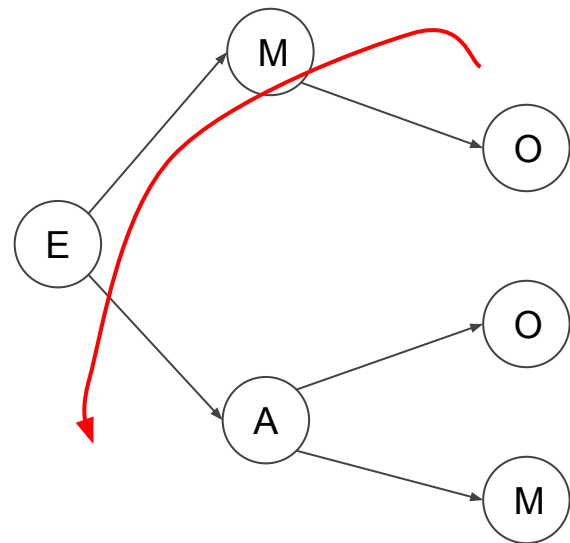
# Árboles

```
data Objeto = Armadura | Escudo  
            | Maza      | Oro  
  
data Dungeon = Armario  
              | Habitacion Objeto  
              Dungeon Dungeon
```

■ Tipos algebraicos recursivos: árboles

■ Los gráficos se suelen usar para facilitar la lectura

```
d4 = Habitacion Escudo  
    (Habitacion Maza  
     Armario  
     (Habitacion Oro  Armario Armario)  
    )  
    (Habitacion Armadura  
     (Habitacion Oro  Armario Armario)  
     (Habitacion Maza Armario Armario)  
    )
```



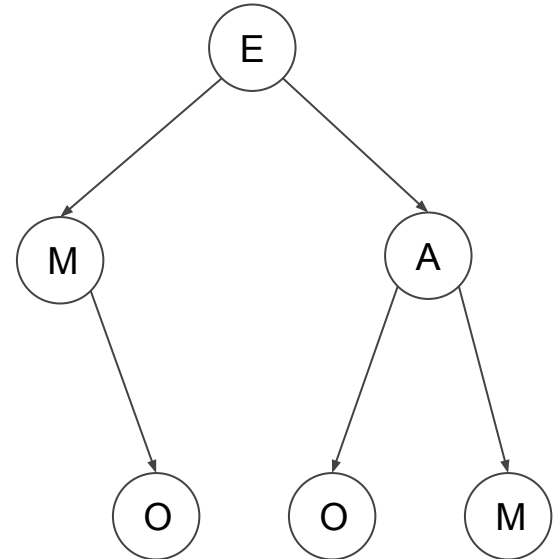
# Árboles

```
data Objeto = Armadura | Escudo
            | Maza      | Oro

data Dungeon = Armario
            | Habitacion Objeto
            | Dungeon Dungeon
```

- Tipos algebraicos recursivos: árboles
- Los gráficos se suelen usar para facilitar la lectura

```
d4 = Habitacion Escudo
    (Habitacion Maza
     Armario
     (Habitacion Oro  Armario Armario)
    )
    (Habitacion Armadura
     (Habitacion Oro  Armario Armario)
     (Habitacion Maza Armario Armario)
    )
```



# Árboles

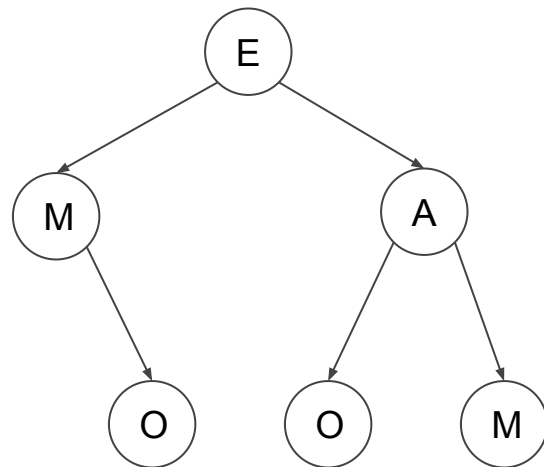
```
data Objeto = Armadura | Escudo
            | Maza      | Oro

data Dungeon = Armario
            | Habitacion Objeto
            | Dungeon Dungeon
```

■ Tipos algebraicos recursivos: árboles

■ Los gráficos se suelen usar para facilitar la lectura

```
d4 = Habitacion Escudo
    (Habitacion Maza
     Armario
     (Habitacion Oro  Armario Armario)
    )
    (Habitacion Armadura
     (Habitacion Oro  Armario Armario)
     (Habitacion Maza Armario Armario)
    )
```



# Árboles

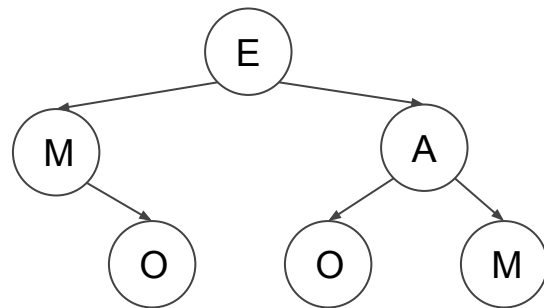
```
data Objeto = Armadura | Escudo
            | Maza      | Oro

data Dungeon = Armario
            | Habitacion Objeto
            | Dungeon Dungeon
```

■ Tipos algebraicos recursivos: árboles

■ Los gráficos se suelen usar para facilitar la lectura

```
d4 = Habitacion Escudo
    (Habitacion Maza
     Armario
     (Habitacion Oro  Armario Armario)
    )
    (Habitacion Armadura
     (Habitacion Oro  Armario Armario)
     (Habitacion Maza Armario Armario)
    )
```



# Árboles

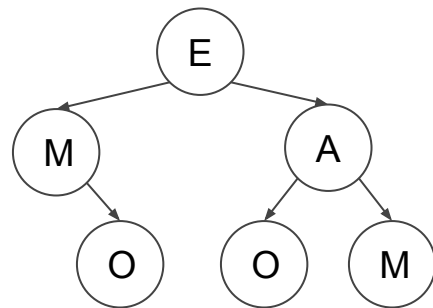
```
data Objeto = Armadura | Escudo
            | Maza      | Oro

data Dungeon = Armario
            | Habitacion Objeto
            | Dungeon Dungeon
```

■ Tipos algebraicos recursivos: árboles

■ Los gráficos se suelen usar para facilitar la lectura

```
d4 = Habitacion Escudo
    (Habitacion Maza
     Armario
     (Habitacion Oro Armario Armario)
    )
    (Habitacion Armadura
     (Habitacion Oro Armario Armario)
     (Habitacion Maza Armario Armario)
    )
```



# Árboles

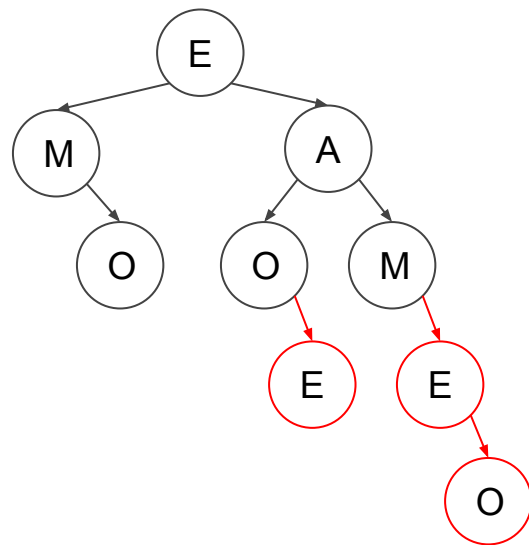
```
data Objeto = Armadura | Escudo
            | Maza      | Oro

data Dungeon = Armario
            | Habitacion Objeto
            | Dungeon Dungeon
```

■ Tipos algebraicos recursivos: árboles

■ Los gráficos se suelen usar para facilitar la lectura

```
d4 = Habitacion Escudo
    (Habitacion Maza Armario
     (Habitacion Oro Armario Armario)
    )
    (Habitacion Armadura
     (Habitacion Oro Armario Armario)
     (Habitacion Maza Armario Armario)
    )
```





# Árboles

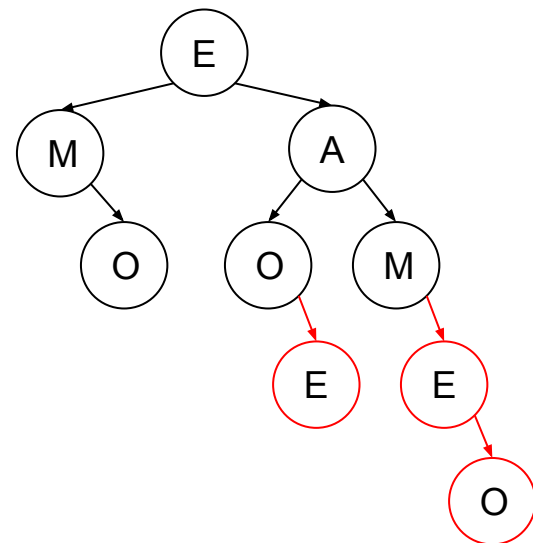
```
data Objeto = Armadura | Escudo
            | Maza      | Oro

data Dungeon = Armario
            | Habitacion Objeto
            | Dungeon Dungeon
```

■ Tipos algebraicos recursivos: árboles

■ Los gráficos se suelen usar para facilitar la lectura

```
d5 = Habitacion Escudo
    (Habitacion Maza Armario
     (Habitacion Oro Armario Armario)
    )
    (Habitacion Armadura
     (Habitacion Oro Armario
      (Habitacion Escudo Armario Armario))
     (Habitacion Maza Armario
      (Habitacion Escudo Armario
       (Habitacion Oro Armario Armario))
     )
    )
  )
```



# Árboles

```
data Objeto = Armadura | Escudo
            | Maza      | Oro

data Dungeon = Armario
            | Habitacion Objeto
            | Dungeon Dungeon
```

- Tipos algebraicos recursivos: árboles
  - Funciones recursivas estructurales
    - Un caso por cada constructor
    - La misma función en las partes recursivas

```
f :: Dungeon -> b
f Armario          = ...
f (Habitacion obj d1 d2) = ... obj ... f d1 ... f d2 ...
```

# Árboles

```
data Objeto = Armadura | Escudo
            | Maza      | Oro

data Dungeon = Armario
            | Habitacion Objeto
            | Dungeon Dungeon
```

- Tipos algebraicos recursivos: árboles
- Funciones recursivas estructurales
  - Primero se plantea la recursión

```
cantidadDeOro :: Dungeon -> Int
cantidadDeOro Armario = ...
cantidadDeOro (Habitacion obj d1 d2) =
    ... obj ... cantidadDeOro d1 ... cantidadDeOro d2 ...
```

# Árboles

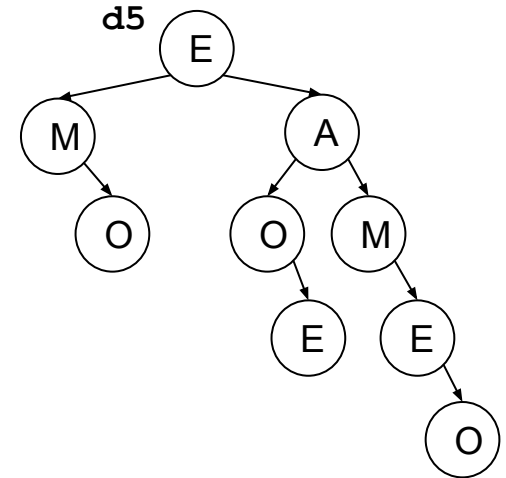
```
data Objeto = Armadura | Escudo
            | Maza      | Oro

data Dungeon = Armario
              | Habitacion Objeto
              | Dungeon Dungeon

cantidadDeOro :: Dungeon -> Int
cantidadDeOro Armario = ...
cantidadDeOro (Habitacion obj d1 d2) =
    ... obj ... cantidadDeOro d1 ...
    ... cantidadDeOro d2 ...
```

- Tipos algebraicos recursivos: árboles
- Funciones recursivas estructurales
  - Se prueba con un ejemplo

`cantidadDeOro d5`



# Árboles

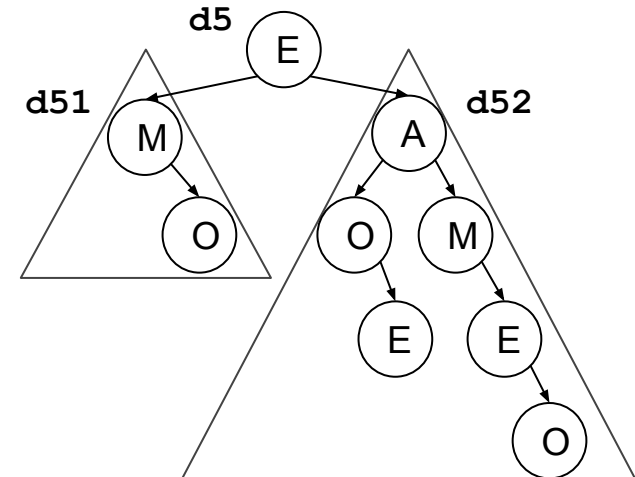
```
data Objeto = Armadura | Escudo
            | Maza      | Oro

data Dungeon = Armario
              | Habitacion Objeto
              | Dungeon Dungeon

cantidadDeOro :: Dungeon -> Int
cantidadDeOro Armario = ...
cantidadDeOro (Habitacion obj d1 d2) =
    ... obj ... cantidadDeOro d1 ...
    ... cantidadDeOro d2 ...
```

- Tipos algebraicos recursivos: árboles
- Funciones recursivas estructurales
  - Se prueba con un ejemplo

```
cantidadDeOro d5
=
    ... Escudo ... cantidadDeOro d51 ...
    ... cantidadDeOro d52 ...
```



# Árboles

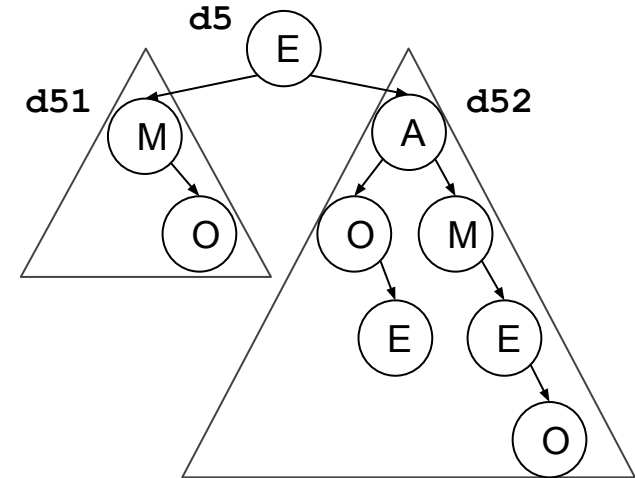
```
data Objeto = Armadura | Escudo
            | Maza      | Oro

data Dungeon = Armario
              | Habitacion Objeto
              | Dungeon Dungeon

cantidadDeOro :: Dungeon -> Int
cantidadDeOro Armario = ...
cantidadDeOro (Habitacion obj d1 d2) =
    ... obj ... cantidadDeOro d1 ...
    ... cantidadDeOro d2 ...
```

- Tipos algebraicos recursivos: árboles
- Funciones recursivas estructurales
  - Se prueba con un ejemplo

```
cantidadDeOro d5
=
    ... Escudo ... cantidadDeOro d51 ...
    ... cantidadDeOro d52 ...
=
    ... Escudo ... 1 ...
    ... 2 ...
```



# Árboles

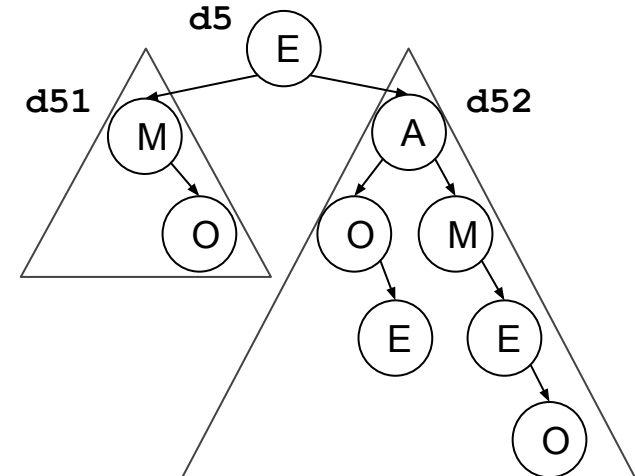
```
data Objeto = Armadura | Escudo
            | Maza      | Oro

data Dungeon = Armario
              | Habitacion Objeto
              | Dungeon Dungeon

cantidadDeOro :: Dungeon -> Int
cantidadDeOro Armario = ...
cantidadDeOro (Habitacion obj d1 d2) =
    ... obj ... cantidadDeOro d1 ...
    ... cantidadDeOro d2 ...
```

- Tipos algebraicos recursivos: árboles
- Funciones recursivas estructurales
  - Se prueba con un ejemplo

```
cantidadDeOro d5
=
    ... Escudo ... cantidadDeOro d51 ...
    ... cantidadDeOro d52 ...
=
    ... Escudo ... 1 ...
    ... 2 ...
=
    3
```



# Árboles

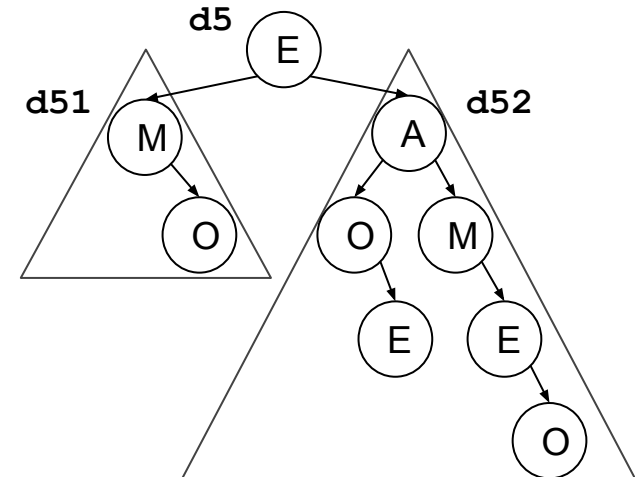
```
data Objeto = Armadura | Escudo
            | Maza      | Oro

data Dungeon = Armario
              | Habitacion Objeto
              | Dungeon Dungeon

cantidadDeOro :: Dungeon -> Int
cantidadDeOro Armario = ...
cantidadDeOro (Habitacion obj d1 d2) =
    ... obj ... cantidadDeOro d1 ...
    ... cantidadDeOro d2 ...
```

- Tipos algebraicos recursivos: árboles
- Funciones recursivas estructurales
  - Se prueba con un ejemplo

```
cantidadDeOro d5
=
  unoSiEsOro Escudo + cantidadDeOro d51
                    + cantidadDeOro d52
=
  unoSiEsOro Escudo + 1
                    + 2
=
  3
```





# Árboles

- Tipos algebraicos recursivos: árboles

- Funciones recursivas estructurales

- Luego se resuelve el caso recursivo

(agregando lo que falte, quizás con funciones auxiliares)

```
cantidadDeOro :: Dungeon -> Int
cantidadDeOro Armario = ...
cantidadDeOro (Habitacion obj d1 d2) =
    unoSiEsOro obj + cantidadDeOro d1
    + cantidadDeOro d2
```

```
unoSiEsOro :: Objeto -> Int
unoSiEsOro ...
```

```
data Objeto = Armadura | Escudo
            | Maza      | Oro

data Dungeon = Armario
            | Habitacion Objeto
                Dungeon Dungeon

cantidadDeOro :: Dungeon -> Int
cantidadDeOro Armario = ...
cantidadDeOro (Habitacion obj d1 d2) =
    unoSiEsDeOro obj + cantidadDeOro d1
    + cantidadDeOro d2

unoSiEsDeOro :: Objeto -> Int
```

# Árboles

- Tipos algebraicos recursivos: árboles

- Funciones recursivas estructurales

- Luego se resuelve el caso recursivo

(agregando lo que falte, quizás con funciones auxiliares)

```
cantidadDeOro :: Dungeon -> Int
cantidadDeOro Armario = ...
cantidadDeOro (Habitacion obj d1 d2) =
    unoSiEsOro obj + cantidadDeOro d1
    + cantidadDeOro d2
```

```
unoSiEsOro :: Objeto -> Int
unoSiEsOro Oro = 1
unoSiEsOro _ = 0
```

```
data Objeto = Armadura | Escudo
            | Maza      | Oro

data Dungeon = Armario
            | Habitacion Objeto
            | Dungeon Dungeon

cantidadDeOro :: Dungeon -> Int
cantidadDeOro Armario = ...
cantidadDeOro (Habitacion obj d1 d2) =
    unoSiEsDeOro obj + cantidadDeOro d1
    + cantidadDeOro d2

unoSiEsDeOro :: Objeto -> Int
```

# Árboles

- Tipos algebraicos recursivos: árboles

- Funciones recursivas estructurales

- Se completa con el caso base

```
data Objeto = Armadura | Escudo
            | Maza      | Oro

data Dungeon = Armario
              | Habitacion Objeto
                Dungeon Dungeon

cantidadDeOro :: Dungeon -> Int
cantidadDeOro Armario = 0
cantidadDeOro (Habitacion obj d1 d2) =
    unoSiEsDeOro obj + cantidadDeOro d1
    + cantidadDeOro d2

unoSiEsDeOro :: Objeto -> Int
```

```
cantidadDeOro :: Dungeon -> Int
cantidadDeOro Armario = 0
cantidadDeOro (Habitacion obj d1 d2) =
    unoSiEsOro obj + cantidadDeOro d1
    + cantidadDeOro d2
```

```
unoSiEsOro :: Objeto -> Int
unoSiEsOro Oro = 1
unoSiEsOro _ = 0
```

# Árboles



## Más funciones sobre Dungeons

```
profundidad :: Dungeon -> Int
    -- El camino más largo desde el inicio

cambiarMazasPorOro :: Dungeon -> Dungeon
    -- Quita todas las mazas, y deja Oro en su lugar

objetos :: Dungeon -> [Objeto]
    -- Todos los objetos del dungeon

objsDelCaminoMasLargo :: Dungeon -> [Objeto]
    -- Los objetos que están solamente en el camino más largo
```

```
data Objeto = Armadura | Escudo
            | Maza      | Oro

data Dungeon = Armario
            | Habitacion Objeto
            | Dungeon Dungeon

cantidadDeOro :: Dungeon -> Int
cantidadDeOro Armario = 0
cantidadDeOro (Habitacion obj d1 d2) =
    unoSiEsDeOro obj + cantidadDeOro d1
    + cantidadDeOro d2

unoSiEsDeOro :: Objeto -> Int
```

# Árboles

```
data Objeto = Armadura | Escudo
             | Maza      | Oro

data Dungeon = Armario
              | Habitacion Objeto
              | Dungeon Dungeon
```

## ■ Tipos algebraicos recursivos: árboles

### ■ Un ejemplo más

```
objsDelCaminoMasLargo :: Dungeon -> [Objeto]
objsDelCaminoMasLargo Armario = ...
objsDelCaminoMasLargo (Habitacion obj d1 d2) =
    ... obj ... objsDelCaminoMasLargo d1
    ... objsDelCaminoMasLargo d2 ...
```

# Árboles

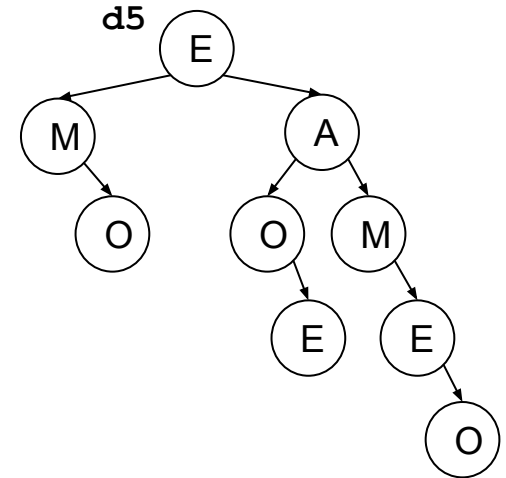
- Tipos algebraicos recursivos: árboles
- Un ejemplo más

objsDelCaminoMasLargo d5

```
data Objeto = Armadura | Escudo
            | Maza      | Oro

data Dungeon = Armario
              | Habitacion Objeto
                Dungeon Dungeon

cantidadDeOro :: Dungeon -> Int
cantidadDeOro Armario = ...
cantidadDeOro (Habitacion obj d1 d2) =
    ... obj ... cantidadDeOro d1 ...
    ... cantidadDeOro d2 ...
```



# Árboles

## Tipos algebraicos recursivos: árboles

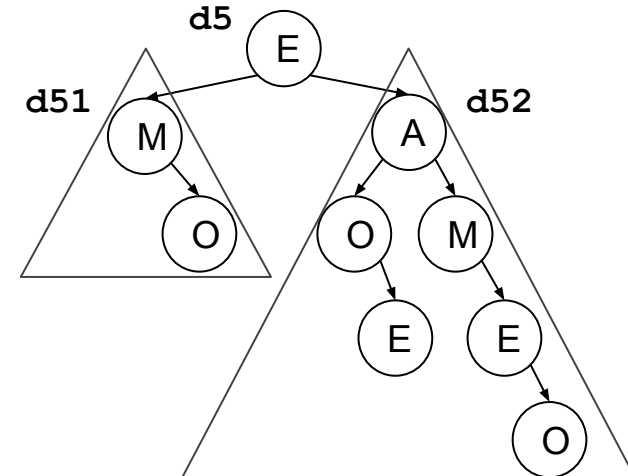
### Un ejemplo más

```
objsDelCaminoMasLargo d5
=
... Escudo ... objsDelCaminoMasLargo d51 ...
... objsDelCaminoMasLargo d52 ...
```

```
data Objeto = Armadura | Escudo
            | Maza      | Oro

data Dungeon = Armario
              | Habitacion Objeto
              | Dungeon Dungeon

cantidadDeOro :: Dungeon -> Int
cantidadDeOro Armario = ...
cantidadDeOro (Habitacion obj d1 d2) =
    ... obj ... cantidadDeOro d1 ...
    ... cantidadDeOro d2 ...
```



# Árboles

## Tipos algebraicos recursivos: árboles

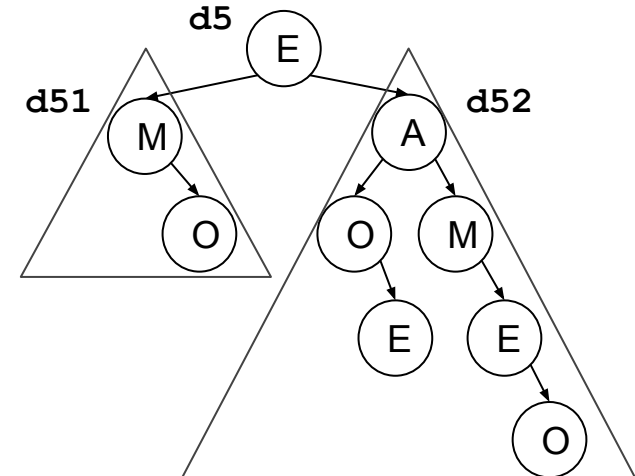
### Un ejemplo más

```
data Objeto = Armadura | Escudo
            | Maza      | Oro

data Dungeon = Armario
              | Habitacion Objeto
              | Dungeon Dungeon

cantidadDeOro :: Dungeon -> Int
cantidadDeOro Armario = ...
cantidadDeOro (Habitacion obj d1 d2) =
    ... obj ... cantidadDeOro d1 ...
    ... cantidadDeOro d2 ...
```

```
objsDelCaminoMasLargo d5
=
    ... Escudo ... objsDelCaminoMasLargo d51 ...
    ... objsDelCaminoMasLargo d52 ...
=
    ... Escudo ... [Maza,Oro] ...
    ... [Armadura,Maza,Escudo,Oro] ...
```





# Árboles

## Tipos algebraicos recursivos: árboles

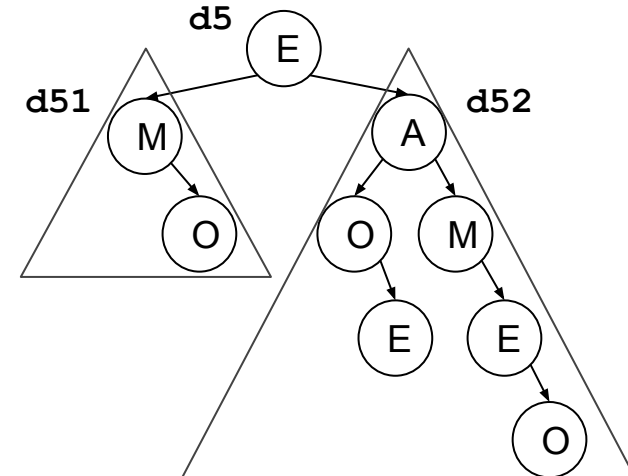
### Un ejemplo más

```
objsDelCaminoMasLargo d5
=
... Escudo ... objsDelCaminoMasLargo d51 ...
... objsDelCaminoMasLargo d52 ...
=
... Escudo ... [Maza,Oro] ...
... [Armadura,Maza,Escudo,Oro] ...
=
Escudo : [Armadura,Maza,Escudo,Oro]
```

```
data Objeto = Armadura | Escudo
            | Maza      | Oro

data Dungeon = Armario
              | Habitacion Objeto
              | Dungeon Dungeon

cantidadDeOro :: Dungeon -> Int
cantidadDeOro Armario = ...
cantidadDeOro (Habitacion obj d1 d2) =
    ... obj ... cantidadDeOro d1 ...
    ... cantidadDeOro d2 ...
```



# Árboles

```
data Objeto = Armadura | Escudo
             | Maza      | Oro

data Dungeon = Armario
              | Habitacion Objeto
              | Dungeon Dungeon
```

## ■ Tipos algebraicos recursivos: árboles

### ■ Un ejemplo más

```
objsDelCaminoMasLargo :: Dungeon -> [Objeto]
objsDelCaminoMasLargo Armario = ...
objsDelCaminoMasLargo (Habitacion obj d1 d2) =
    obj : elegirEntre (objsDelCaminoMasLargo d1)
                     (objsDelCaminoMasLargo d2)

elegirEntre :: [Objeto] -> [Objeto] -> [Objeto]
elegirEntre objs1 objs2 = ...
```

# Árboles

```
data Objeto = Armadura | Escudo
             | Maza      | Oro

data Dungeon = Armario
              | Habitacion Objeto
              | Dungeon Dungeon
```

## ■ Tipos algebraicos recursivos: árboles

### ■ Un ejemplo más

```
objsDelCaminoMasLargo :: Dungeon -> [Objeto]
objsDelCaminoMasLargo Armario = ...
objsDelCaminoMasLargo (Habitacion obj d1 d2) =
    obj : elegirEntre (objsDelCaminoMasLargo d1)
                     (objsDelCaminoMasLargo d2)

elegirEntre :: [Objeto] -> [Objeto] -> [Objeto]
elegirEntre objs1 objs2 = if length objs1 > length objs2
                          then objs1
                          else objs2
```

# Árboles

```
data Objeto = Armadura | Escudo
             | Maza      | Oro

data Dungeon = Armario
              | Habitacion Objeto
              | Dungeon Dungeon
```

## Tipos algebraicos recursivos: árboles

### Un ejemplo más

```
objsDelCaminoMasLargo :: Dungeon -> [Objeto]
objsDelCaminoMasLargo Armario           = []
objsDelCaminoMasLargo (Habitacion obj d1 d2) =
    obj : elegirEntre (objsDelCaminoMasLargo d1)
                      (objsDelCaminoMasLargo d2)

elegirEntre :: [Objeto] -> [Objeto] -> [Objeto]
elegirEntre objs1 objs2 = if length objs1 > length objs2
                           then objs1
                           else objs2
```

# Árboles

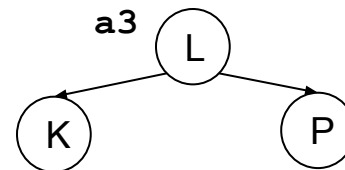
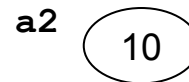
- ❏ ¿Cómo serían si solamente queremos la estructura?
- ❏ Se conoce como *árbol binario*
- ❏ La recursión sigue la estructura

```
data Tree a = EmptyT  
            | NodeT a (Tree a) (Tree a)
```

```
a1 = EmptyT
```

```
a2 = NodeT 10 EmptyT EmptyT
```

```
a3 = NodeT 'L' (Node 'K' EmptyT EmptyT)  
             (Node 'P' EmptyT EmptyT)
```



# Árboles

```
data Tree a = EmptyT
             | NodeT a (Tree a) (Tree a)
```

## Funciones sobre **Trees**

```
f :: Tree a -> b
f EmptyT          = ...
f (NodeT x t1 t2) = ... x ... f t1 ... f t2 ...
```

```
armarDungeon :: Tree Objeto -> Dungeon
-- Arma un dungeon donde cada nodo representa una
--   Habitación y los árboles vacíos representa Armarios
```

# Resumen

# Resumen

- Tipos algebraicos recursivos
  - Sumas con algún constructor con el mismo tipo como argumento
  - Se utilizan *las mismas técnicas* que con otros tipos algebraicos
  - Recursión estructural
    - Una ecuación por cada constructor
    - La misma función en las partes recursivas
    - Solo hace falta pensar cómo agregar lo que falta