Correctitud en TADs

Repaso

- Un TAD define el **qué**. Tiene estado y operaciones descriptas mediante pre y post condición, en lógica

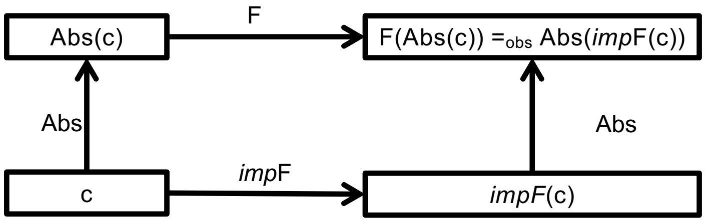
- Una implementación define el **cómo**. Tiene un estado, invariante de representación, función (o predicado) de abstracción y algoritmos para las operaciones
- Un TAD puede tener muchas implementaciones, según los requerimientos y el contexto de uso (por ejemplo de eficiencia)

Verificación

¿Podemos demostrar que la implementación de un TAD es correcta respecto de la especificación del TAD?

- Si!

Para toda operación *impF* que implementa una operación *F* del TAD y toda instancia *c* de su representación que cumple el invariante de representación, debemos ver que el siguiente diagrama conmuta:



Para cada operación, hay que demostrar que:

1. Conserva el invariante
2. El algoritmo respeta la pre y postcondición del TAD

Vamos a tener que viajar entre los mundos de la implementación y el del TAD usando la función de abstracción...

Conservación del invariante - Ejemplo

TAD punto / Operación `rotar`

```
TAD Punto {
  obs x: float
  obs y: float

proc rotar(p: Punto, d: float)
  requiere true
  asegura p.x = auxRho(old(p)) * cos(auxTheta(old(p))+d)
  asegura p.y = auxRho(old(p)) * sin(auxTheta(old(p))+d)

aux auxTheta(p: Punto): float {
  if p.x = 0 then pi/2 sign(p.y) else arctan(p.y/p.x)}

aux auxRho(p: Punto): float {
  sqrt(p.x ** 2 + p.y ** 2)}
}
```

```
Impl PuntoImpl {
  var rho: float
  var theta float

  pred InvRep(p': PuntoImpl) {
    0 <= p'.theta < 2*pi
  }

  FuncAbs(p': PuntoImpl): Punto {
    Punto p | p.x = p'.rho * cos(p'.theta) &&
              p.y = p'.rho * sin(p'.theta
  }
  proc rotar(p': PuntoImpl, d: float) {
    p'.theta = p'.theta + d;
  }
}
```

Conservación del invariante

```
  InvRep(p')

wp(código, InvRep(p'))

    p'.theta = p'.theta + d;

  InvRep(p')
```

Tenemos que demostrar que
`InvRep(p') ==> wp(código, InvRep(p'))`

Conservación del invariante

```
0 <= p'.theta < 2*pi InvRep(p')
==>??
 0 <= p'.theta+d < 2*pi

  p'.theta = p'.theta + d;

0 <= p'.theta < 2*pi InvRep(p')
```

No es verdad que

```
0 <= p'.theta < 2*pi
==>
0 <= p'.theta+d < 2*pi
```

¡Oops! Llegamos a que, como precondición, tiene que suceder que el valor del ángulo **más** el parámetro de entrada esté en rango ¿Qué hacemos?

- ¿Corregimos la especificación?
- ¿Corregimos el invariante?
- ¿Corregimos el algoritmo?

Conservación del invariante

¡El algoritmo!

```
0 <= p'.theta < 2*pi InvRep(p’)
==>
0 <= (p'.theta+d)%(2*pi) < 2*pi

  p'.theta = p'.theta +  d;

0 <= p'.theta%(2*pi) < 2*pi

  p'.theta = p'.theta % 2*pi;

0 <= p'.theta < 2*pi InvRep(p’)
```

Ahora sí!

`InvRep(p') ==> true`

Correctitud del algoritmo

Pero el código habla de la representación!

Dado un Punto cualquiera **p** y un PuntoImpl **p'** tales que `p == FuncAbs(p')`

Tenemos que probar que `PTad(p, d) ==> wp(código, QTad(p, d))`

```
proc rotar(p: Punto, d: float)
   requiere PTad

   asegura QTad
```

```
proc rotar(p': PuntoImpl, d: float)
   requiere Invrep && Pimpl

   asegura Invrep && Qimpl
```

Dado un Punto cualquiera **p** y un PuntoImpl **p'** tales que `p == FuncAbs(p')` tenemos que probar que

- `(PTad(p, d)) ==> Pimpl(p', d) (1)`
- `(InvRep(p') && Qimpl(p', d)) ==> QTad(p, d) (2)`

Notar que un Pimp canonico seria **PTad(FuncsAbs(p‘),d)** y un Qimp sería **Qtad(FuncsAbs(p‘),d)**

Sumado a que la tripla de Hoare

`{InvRep(p') && Pimpl(p', d)} Código del rotar {InvRep(p') && Qimpl(p', d)} (3)`

Correctitud del algoritmo

```
Impl PuntoImpl {
  var rho: float
  var theta float

  pred InvRep(p': PuntoImpl) {
    0 <= p'.theta < 2*pi
  }

  FuncAbs(p': PuntoImpl): Punto {
    Punto p | p.x = p'.rho * cos(p'.theta) &&
              p.y = p'.rho * sin(p'.theta
  }
  proc rotar(p': PuntoImpl, d: float) {
    p'.theta = (p'.theta + d) %2*pi;
  }
}
```

Recordemos:

Ptad: true
QTad: p.x = auxRho(old(p)) * cos(auxTheta(old(p)+d) && p.y = auxRho(old(p)) * sin(auxTheta(old(p)+d)
Pimpl: true
Qimpl: p'.rho = old(p').rho && p'.theta = old(p').theta + d % 2pi
Código: p'.theta = (p'.theta + d) % (2 * pi)

Entonces

(1) (PTad(p) => Pimpl(p', d)) (True ==> True)

(3) Pimpl {S} Qimpl
wp(p'.theta = (p'.theta + d) % 2*pi, Qimpl) =
   p'.rho = old(p').rho && <u>(p'.theta + d)</u> % 2pi = (old(p').theta + d) % 2pi

¿Pimpl => wp(S, Qimpl)?

Sí, porque al inicio `p' == old(p')`

Correctitud del algoritmo

```
FuncAbs(p': PuntoImpl): Punto {
  Punto p | p.x = p'.rho * cos(p'.theta)  &&
            p.y = p'.rho * sin(p'.theta
}
```

Falta

`(2) InvRep(p’) && Qimpl(p') => Qtad(p)`

`Qimpl``: p'.rho = old(p').rho &&  p'.theta = (old(p').theta + d)  % 2*pi`

`Qtad``: p.x = auxRho(old(p))*cos(auxTheta(old(p)+d)) && p.y = auxRho(old(p))*sin(auxTheta(old(p)+d))`

Si `p = FuncAbs(p')` entonces vale que: `auxRho(p) == p'.rho , auxTheta(p) == p'.theta`

Si reemplazamos `auxRho` y `auxTheta` en `QImp`:

`Qimpl``: auxRho(p) = auxRho(old(p)) &&  auxTheta(p) = (auxTheta(old(p)) + d) % 2*pi`

Asumimos `Qimpl` cierto, queremos ver que `Qtad` es cierto y reemplazamos `auxRho`, `auxTheta`

`Qtad``: p.x = auxRho(p)*cos(auxTheta(p)) && p.y = auxRho(p)*sin(auxTheta(p))`

esto vale porque básicamente es la conversión de cartesiana a polar y nuevamente a cartesiana.