Clase 03
Razonamiento ecuacional
Inducción estructural

¿Cómo mostramos que un programa hace lo que tiene que hacer?
Prueba formal de propiedades: Demostramos que ciertas expresiones son equivalentes para probar que un algoritmo es correcto con respecto a otro y para razonar sobre optimizaciones y alternativas.

Hipótesis de trabajo
1. Trabajamos con estructuras de datos **finitas**.
2. Trabajamos con **funciones totales**.
3. El programa **no depende del orden** de las ecuaciones.

Igualdades por definición

Principio de reemplazo ⭐
Sea `e1 = e2` una ecuación del programa. Las siguientes operaciones preservan la igualdad de expresiones:
1. Reemplazar *cualquier instancia* de `e1` por `e2`.
2. Reemplazar *cualquier instancia* de `e2` por `e1`.

Si la igualdad se puede demostrar usando sólo el principio de reemplazo decimos que la igualdad vale por definición.

Inducción estructural

Principio de inducción estructural ⭐
Sea un tipo inductivo:

```
data T = CBase₁ <parámetros>
         ...
       | CBaseₙ <parámetros>
       | CRecursivo₁ <parámetros>
         ...
       | CRecursivoₙ <parámetros>
```

Sea $\mathcal{P}$ una propiedad acerca de las expresiones tipo `T` tal que:
- $\mathcal{P}$ vale sobre todos los constructores base de `T`,
- $\mathcal{P}$ vale sobre todos los constructores recursivos de `T`,
  asumiendo como hipótesis inductiva que vale ara los parámetros de tipo `T`,

entonces $\forall x :: \mathrm{T}.\mathcal{P}(x)$.

Principio de inducción sobre booleanos
Si $\mathcal{P}(\mathrm{True})$ y $\mathcal{P}(\mathrm{False})$ entonces $\forall x :: \mathrm{Bool}.\ \mathcal{P}(x)$.

Principio de inducción sobre pares
Si $\forall x :: \mathrm{a}.\ \forall y :: \mathrm{b}.\ \mathcal{P}((x, y))$
entonces $\forall p :: (\mathrm{a}, \mathrm{b}).\ \mathcal{P}(p)$.

Principio de inducción sobre naturales
Si $\mathcal{P}(\mathrm{Zero})$ y $\forall \mathrm{n} :: \mathrm{Nat}.(\mathcal{P}(\mathrm{n}) \Rightarrow \mathcal{P}(\mathrm{Suc\ n}))$,
entonces $\forall \mathrm{n} :: \mathrm{Nat}.\ \mathcal{P}(\mathrm{n})$.

> *Principio de inducción sobre listas*
> Sea $\mathcal{P}$ una propiedad sobre expresiones de tipo `[a]` tal que:
> - $\mathcal{P}([\,])$
> - $\forall x :: a.\ \forall xs :: [a].\ (\mathcal{P}(xs) \Rightarrow \mathcal{P}(x : xs))$
>
> Entonces $\forall xs :: [a].\ \mathcal{P}(xs)$.

> *Principio de inducción sobre árboles binarios*
> Sea $\mathcal{P}$ una propiedad sobre expresiones de tipo `AB a` tal que:
> - $\mathcal{P}(\mathrm{Nil})$
> - $\forall i :: \mathrm{AB}\ a.\ \forall r :: a.\ \forall d :: \mathrm{AB}\ a.$
> $(\mathcal{P}(i) \wedge \mathcal{P}(d)) \Rightarrow \mathcal{P}(\mathrm{Bin}\ i\ r\ d)$
>
> Entonces $\forall x :: \mathrm{AB}\ a.\ \mathcal{P}(x)$.

> *Principio de inducción sobre polinomios*
>
> ```haskell
> data Poli a = X
>             | Cte a
>             | Suma (Poli a) (Poli a)
>             | Prod (Poli a) (Poli a)
> ```
>
> Sea $\mathcal{P}$ una propiedad sobre expresiones de tipo `Poli a` tal que:
> - $\mathcal{P}(X)$
> - $\forall k :: a.\ \mathcal{P}(\mathrm{Cte}\ k)$
> - $\forall p :: \mathrm{Poli}\ a.\ \forall q :: \mathrm{Poli}\ a.$
> $((\mathcal{P}(p) \wedge \mathcal{P}(q))) \Rightarrow \mathcal{P}(\mathrm{Suma}\ p\ q)$
> - $\forall p :: \mathrm{Poli}\ a.\ \forall q :: \mathrm{Poli}\ a.$
> $((\mathcal{P}(p) \wedge \mathcal{P}(q))) \Rightarrow \mathcal{P}(\mathrm{Prod}\ p\ q)$
>
> Entonces $\forall x :: \mathrm{Poli}\ a.\ \mathcal{P}(x)$

Lemas de generación

Usando el principio de inducción estructural, se puede probar:

> *Lema de generación para pares*
> Si `p :: (a, b)`, entonces $\exists x :: a.\ \exists y :: b.\ p = (x, y)$.

> *Lema de generación para sumas*
> Si `e :: Either a b`, entonces:
> - o bien $\exists x :: a.\ e = \mathrm{Left}\ x$
> - o bien $\exists y :: b.\ e = \mathrm{Right}\ y$

Extensionalidad

Una equivalencia puede valer dependiendo el punto de vista.

Punto de vista intencional
Dos valores son iguales si están construidos de la misma manera.

Punto de vista extensional
Dos valores son iguales si son indistinguibles al observarlos.

> *Principio de extensionalidad funcional* ⭐
> Sean `f, g :: a -> b`.
> Si $(\forall x :: a.\ f\ x = g\ x)$ entonces f = g

Algunas propiedades utiles para demostraciones

Sea $\forall F :: a \to b. \quad \forall G :: a \to b. \quad \forall Y :: b. \quad \forall Z :: a.$

`F = G` $\Leftrightarrow \forall x :: a.\ F\ x = G\ x$
`F = \x -> Y` $\Leftrightarrow \forall x :: a.\ F\ x = Y$
`(\x -> Y) Z` $\underset{\beta}{=}$ Y reemplazando x por Z
`\x -> F x` $\underset{\eta}{=}$ F

x no puede aparecer libre en F, G ni Z.

Pasos a seguir en una demostración

- Leer la propiedad, entenderla y convencerse de que es verdadera.
- Plantear la propiedad como predicado unario.
- Plantear el esquema de inducción.
- Plantear y resolver el o los caso(s) base.
- Plantear y resolver el o los caso(s) inductivo(s).

Isomorfismos de tipos

Dos tipos de datos son isomorfos cuando representan la misma información, pero escrita de distinta manera.

Decimos que dos tipos de datos A y B son **isomorfos** si:
1. Hay una función f :: A -> B total.
2. Hay una función g :: B -> A total.
3. Se puede demostrar que g . f = id :: A -> A.
4. Se puede demostrar que f . g = id :: B -> B.

Escribimos A $\simeq$ B para indicar que A y B son isomorfos.