Programación Funcional

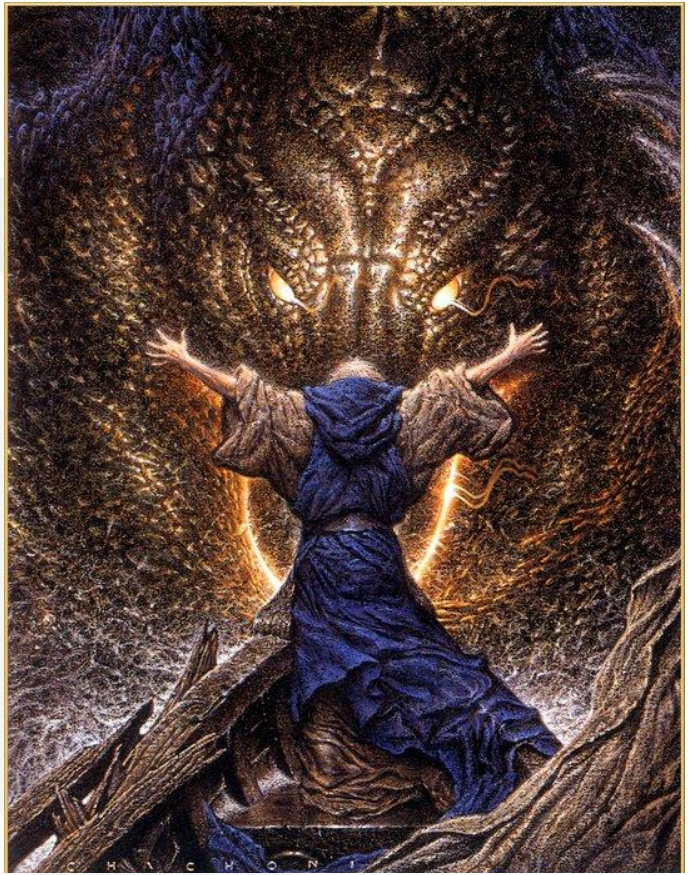
Clases teóricas

por Pablo E. “Fidel” Martínez López

2. Sistemas de tipos

“Ged, acurrucado entre unos matorrales, mojado y melancólico, se preguntaba de qué servía tener poder si una prudencia excesiva impedía utilizarlo, y lamentaba no haber entrado de aprendiz del hechicero del Valle, donde al menos hubiera podido dormir seco.”

Un mago de Terramar
Úrsula K. Le Guin



Motivación de sistemas de tipos

Motivación

- ¿Toda combinación de símbolos es una expresión?

Motivación

- ¿Toda combinación de símbolos es una expresión?
- Consideremos
  - (2+

Motivación

- ¿Toda combinación de símbolos es una expresión?
- Consideremos
  - (2+
    - En este ejemplo, faltan símbolos

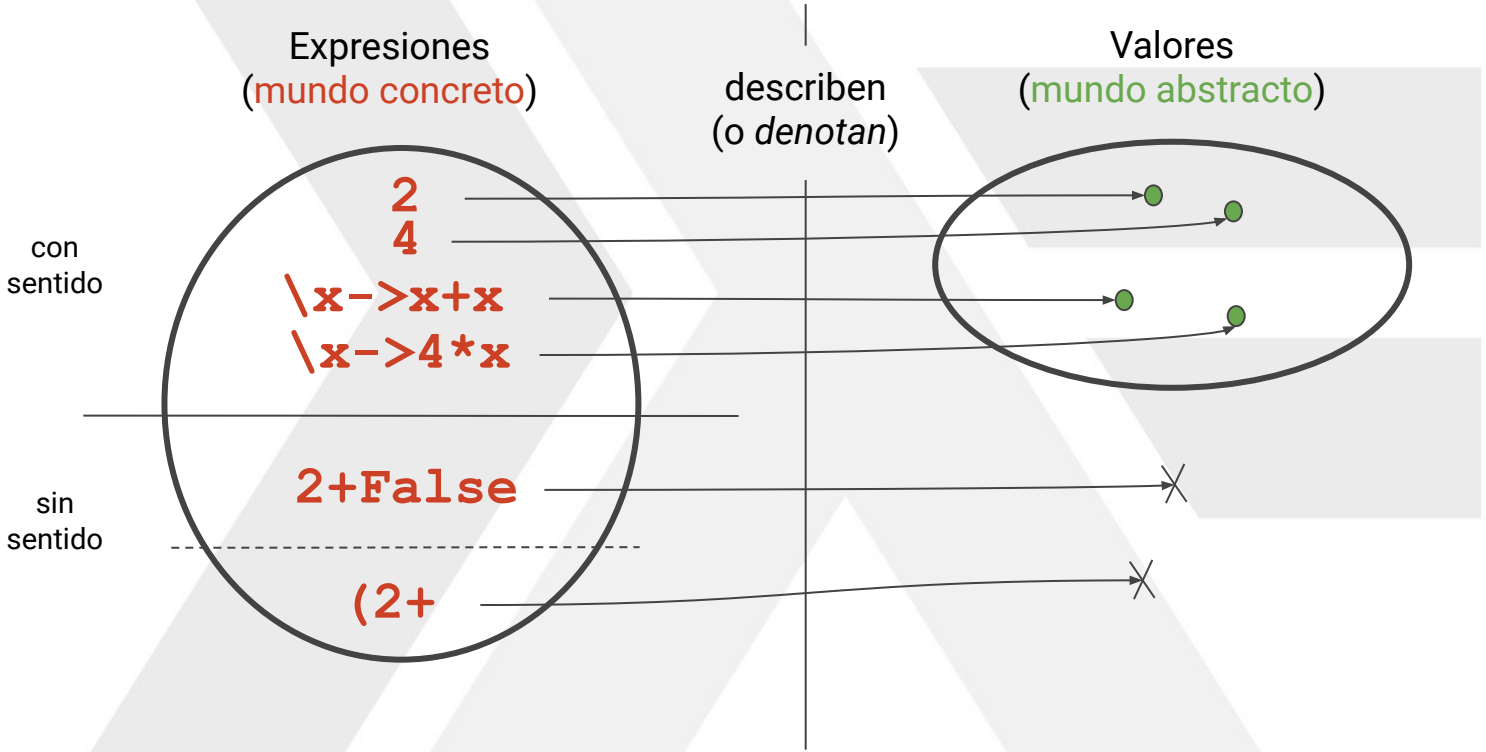
Motivación

- ¿Toda combinación de símbolos es una expresión?
- Consideremos
  - `(2+`
    - En este ejemplo, faltan símbolos
  - `2+False`

Motivación

- ¿Toda combinación de símbolos es una expresión?
- Consideremos
  - `(2+`
    - En este ejemplo, faltan símbolos
  - `2+False`
    - Pero este podría tener significado

Motivación


Expresiones
(mundo concreto)
describen
(o *denotan*)
Valores
(mundo abstracto)
con
sentido
2
4
\x->x+x
\x->4*x
sin
sentido
2+False
(2+

Motivación

- ¿Cómo determinar si una expresión es válida?
  - O sea, si tiene sentido

Motivación

- ¿Cómo determinar si una secuencia de caracteres es una expresión válida?
  - O sea, si es expresión, y si tiene sentido
- Reglas sintácticas
  - BNF u otras formas similares

Motivación

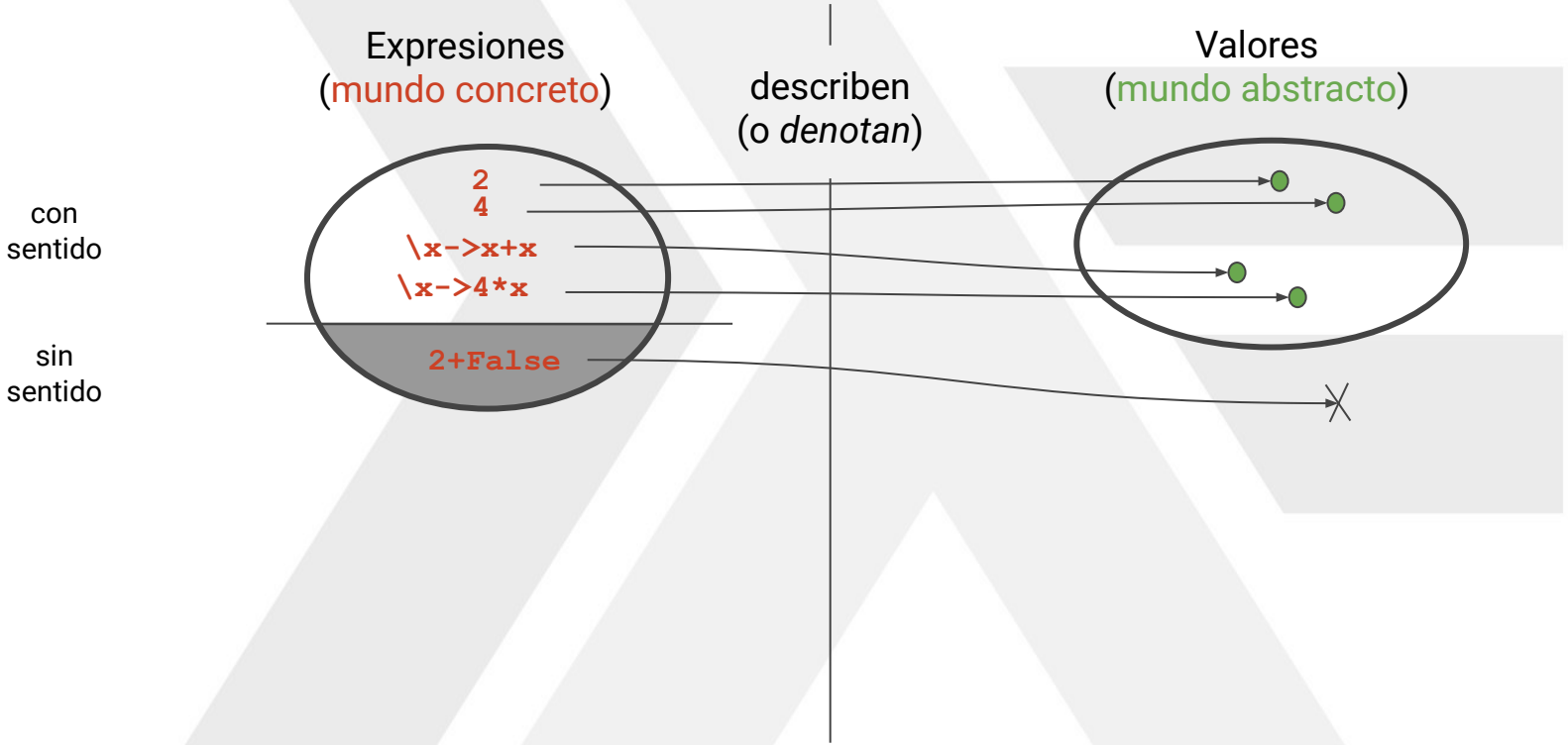
- ¿Cómo determinar si una secuencia de caracteres es una expresión válida?
  - O sea, si es expresión, y si tiene sentido
  - Reglas sintácticas
    - BNF u otras formas similares
  - Reglas de asignación de tipo
    - Gran variedad de sistemas y características

Sistemas de tipos

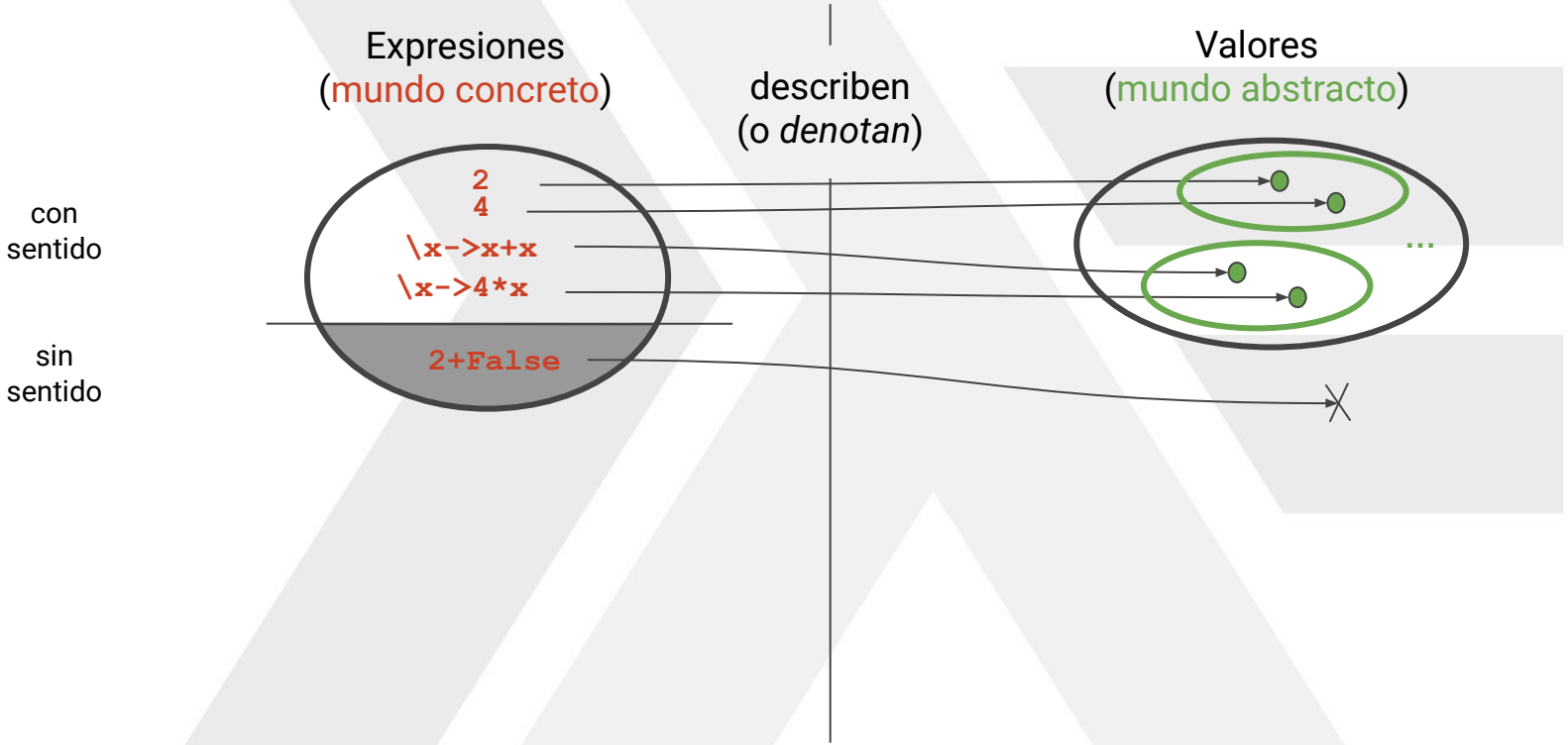
Sistemas de tipos

- Buscamos un mecanismo para clasificar expresiones en base a su sentido
  - Expresiones sin sentido no podrán ser clasificadas
- Es razonable clasificar los valores en conjuntos
  - Precisamos formas de describir estos conjuntos
  - Expresiones de tipo

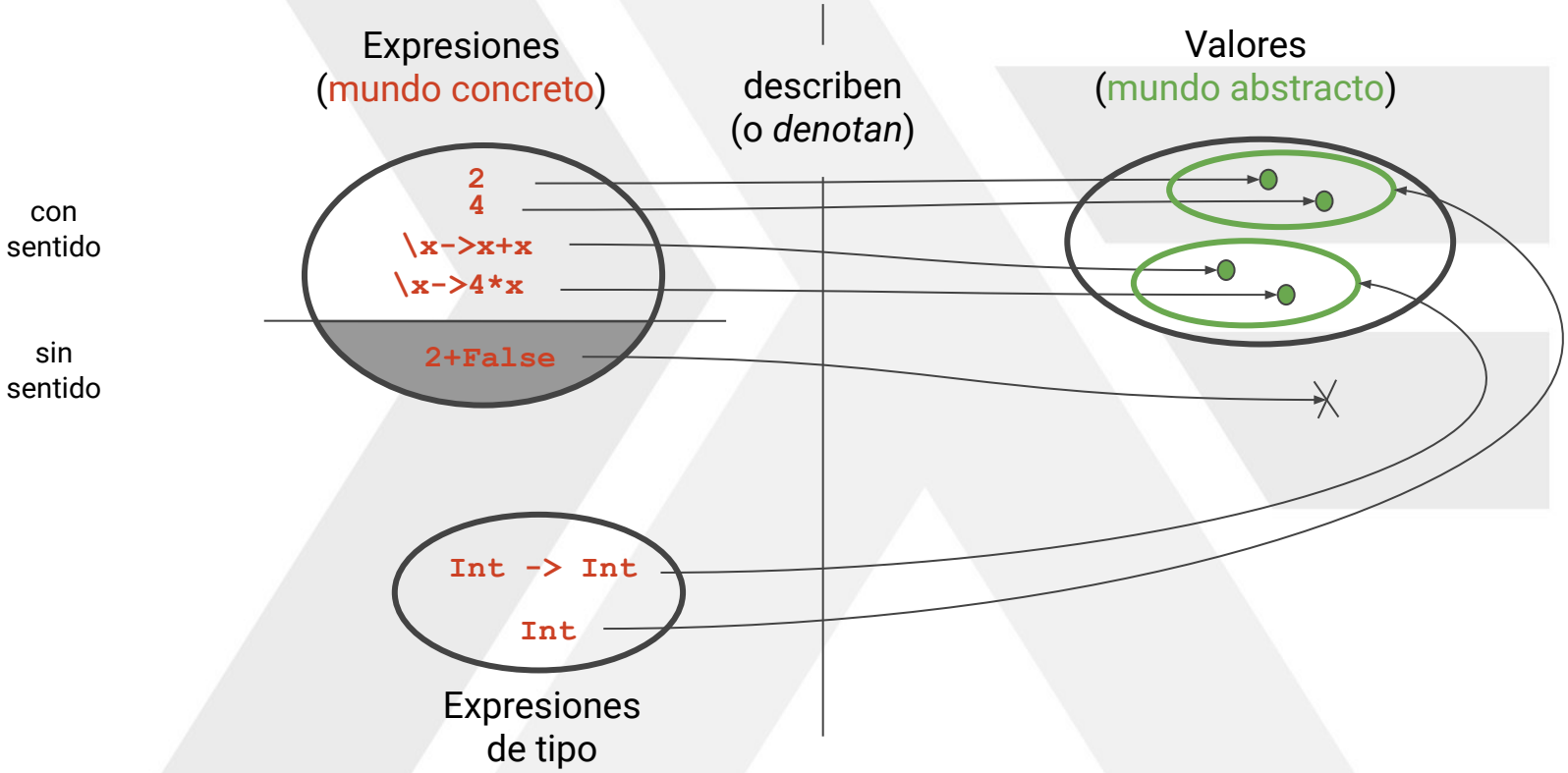
Sistemas de tipos



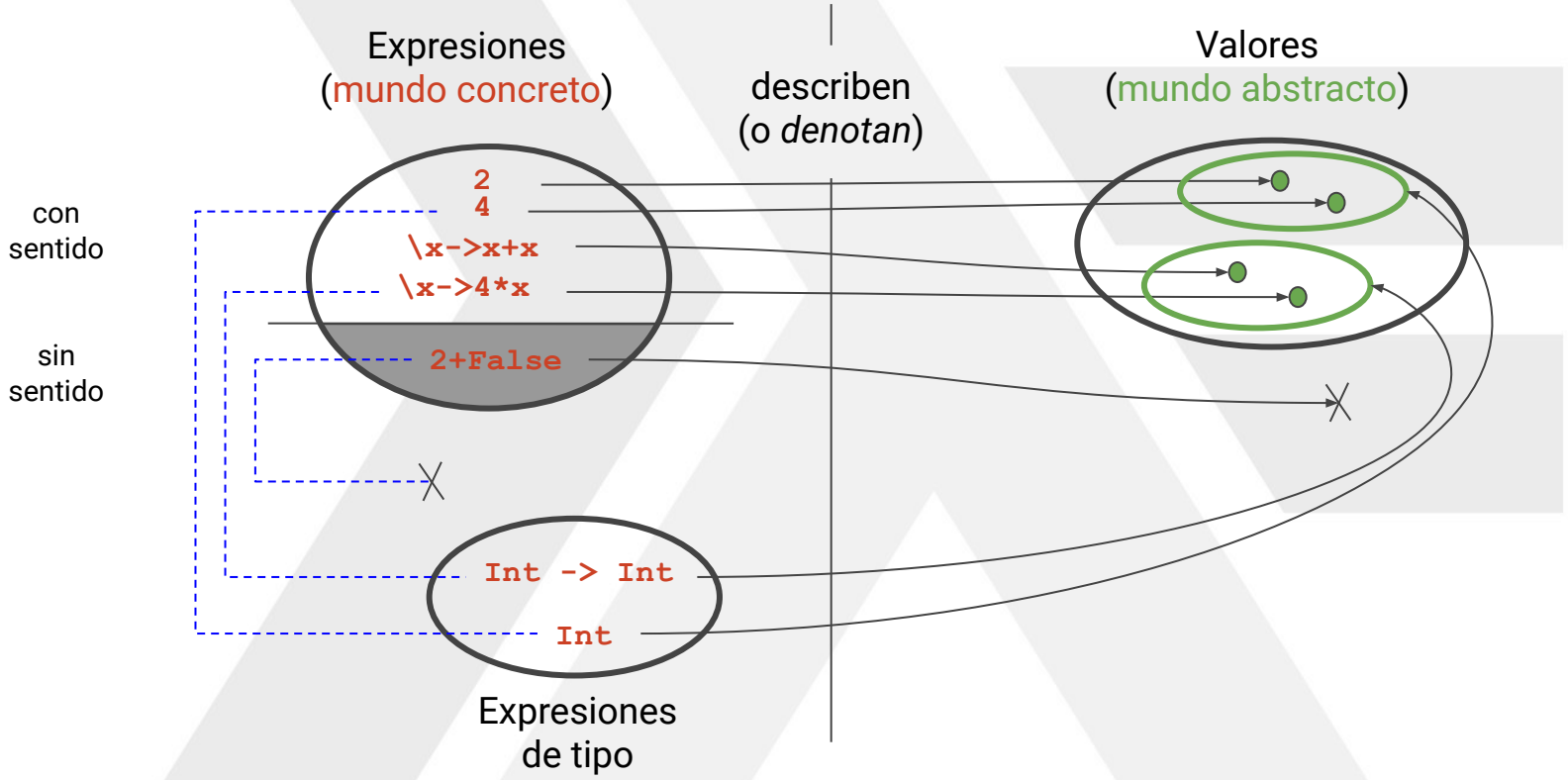
Sistemas de tipos



Sistemas de tipos



Sistemas de tipos



Sistemas de tipos

- Podemos establecer una relación entre expresiones de valores y expresiones de tipo

Sistemas de tipos

- Podemos establecer una relación entre expresiones de valores y expresiones de tipo
- ¿Cómo podemos escribir esta relación?

Sistemas de tipos

- Podemos establecer una relación entre expresiones de valores y expresiones de tipo
- ¿Cómo podemos escribir esta relación?
  - ¡Más símbolos!
    - `2 :: Int`
    - `\x->x+x :: Int -> Int`

Sistemas de tipos

- La notación es **e** : : **A**
- Se lee así: “la expresión **e** tiene tipo **A**”
- Significa que el valor denotado por **e** pertenece al conjunto denotado por **A**
- Las variables **en esta tipografía** son especiales
  - se reemplazan por una expresión válida del conjunto indicado
  - se llaman **metavariables** porque se usan para hablar del lenguaje y NO SON variables del lenguaje

Sistemas de tipos

- ¿Cómo establecer cuándo se cumple esta relación?
  - Sistemas de reglas

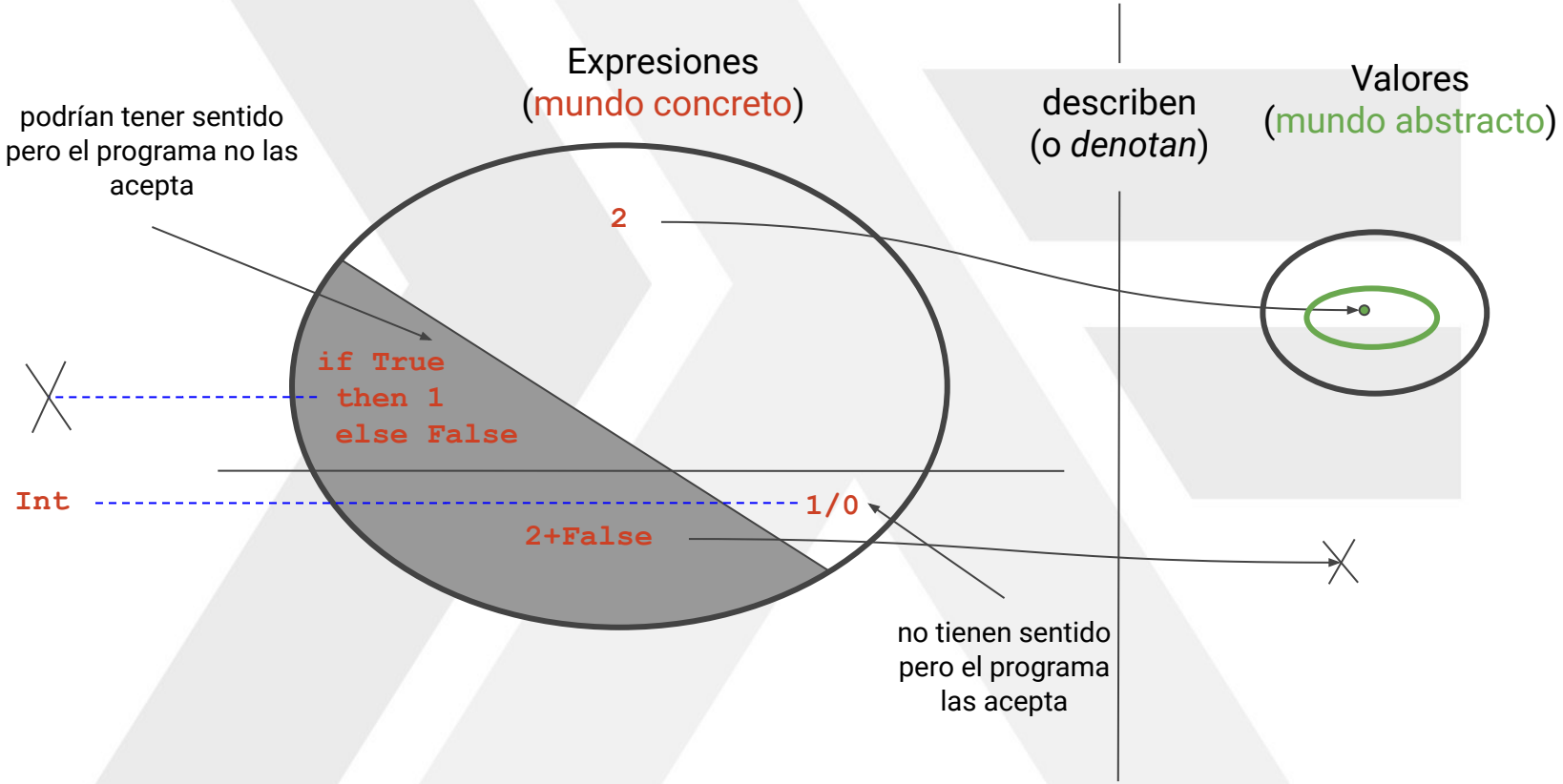
Sistemas de tipos

- ¿Cómo establecer cuándo se cumple esta relación?
  - Sistemas de reglas
  - Si se permite que no haya un programa
  - Si se exige que un programa lo calcule

Sistemas de tipos

- ¿Cómo establecer cuándo se cumple esta relación?
  - Sistemas de reglas
  - Si se permite que no haya un programa
    - Se sacrifica automatización para ganar precisión
    - Sistemas de tipos dependientes (COQ, Agda, etc.)
  - Si se exige que un programa lo calcule
    - Se sacrifica precisión para ganar automatización
    - Vamos a estudiar éstos

Sistemas de tipos calculables

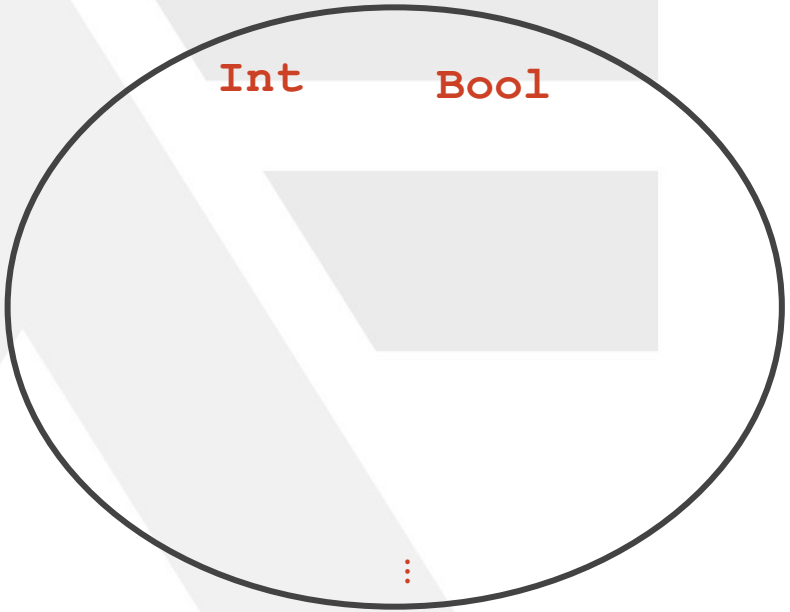


Visiones denotacional y operacional

- **Visión denotacional** **(significado)**
  - = (equivalencia de expresiones)
- **Visión operacional** **(mecánica, ejecutable)**
  - :: (tipado)
  - → (reducción)
- Aceptamos una pérdida de precisión
  - ¿pero cuánta? Depende de la definición del sistema

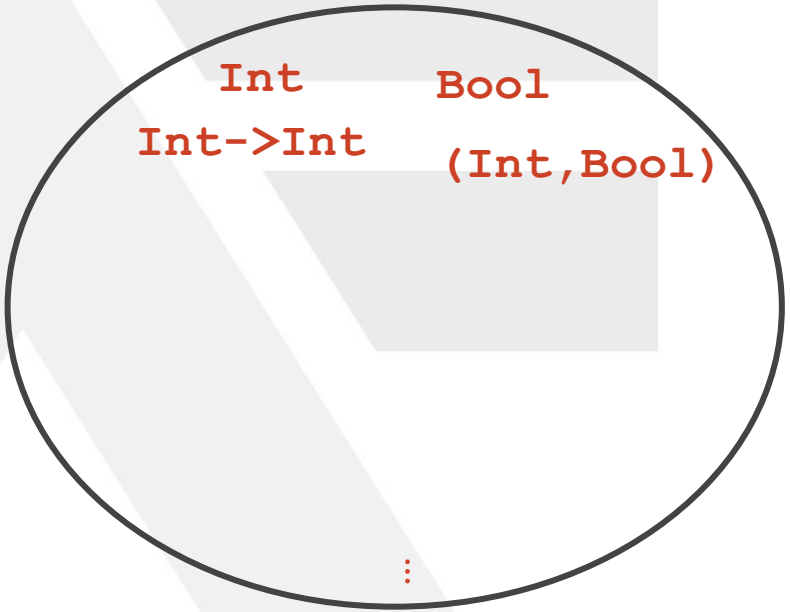
Sistema Hindley-Milner simple

- Se definen tipos (denominados genéricamente **A**, **B**)
  - Básicos
    - `Int`, `Bool`, etc.
  - Compuestos
    - **A** `->` **B**, `(`**A**`,`**B**`)`, etc.



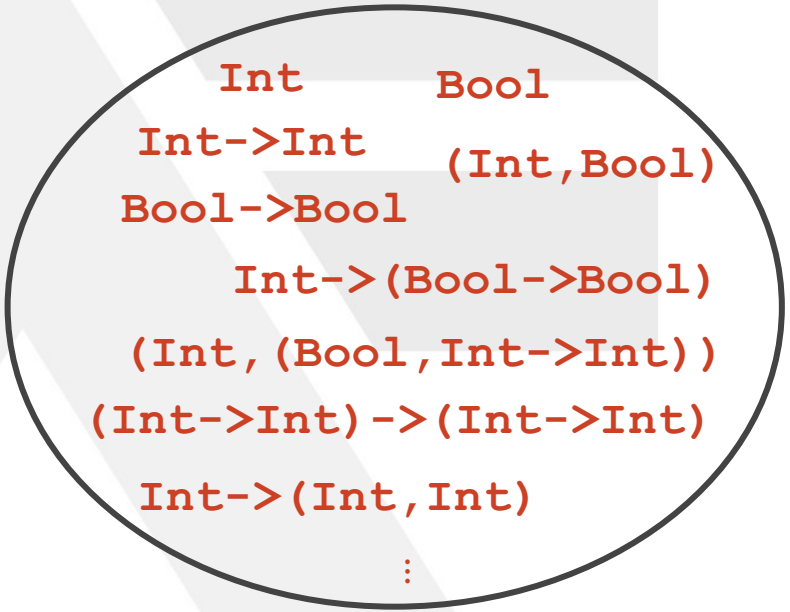
Sistema Hindley-Milner simple

- Se definen tipos (denominados genéricamente **A**, **B**)
  - Básicos
    - `Int`, `Bool`, etc.
  - Compuestos
    - **A** `->` **B**, `(`**A**`,`**B**`)`, etc.



Sistema Hindley-Milner simple

- Se definen tipos (denominados genéricamente **A**, **B**)
  - Básicos
    - `Int`, `Bool`, etc.
  - Compuestos
    - **A** `->` **B**, `(`**A**`,`**B**`)`, etc.



Sistema Hindley-Milner simple

- Se definen reglas para calcular la asignación
  - `n :: Int`
  - `False :: Bool`
  - Si `d = e`
    y `e :: A`
    entonces `d :: A`
  -
    ```
    f :: A -> B
    e :: A
    ---------
    f e :: B
    ```
  -
    ```
    e  :: Int
    e' :: Int
    ---------
    e + e' :: Int
    ```

Sistema Hindley-Milner simple

- Con las reglas se puede calcular el tipo de una expresión a partir de su constitución
- La regla más importante

$$\frac{f :: A \rightarrow B \qquad e :: A}{f\ e :: B}$$

- Dice que en una aplicación, la parte izquierda es una función y la parte derecha tiene el mismo tipo que el tipo del parámetro

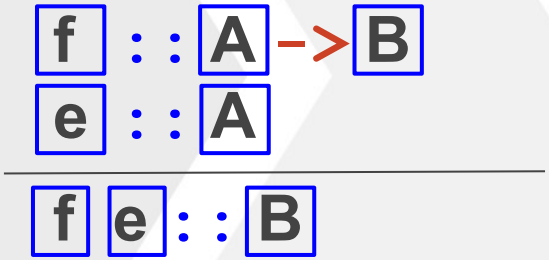
Sistema Hindley-Milner simple

- Las reglas se leen como esquemas a encontrar

$$\frac{f :: A \rightarrow B \qquad e :: A}{f\ e :: B}$$

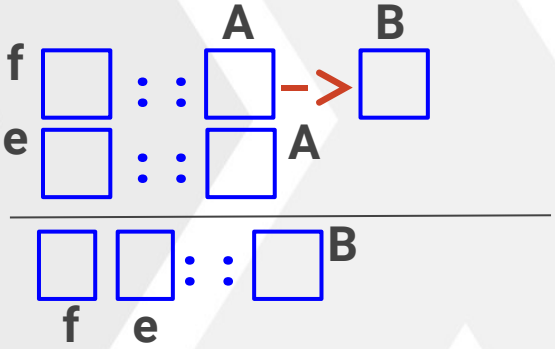
Sistema Hindley-Milner simple

- Las reglas se leen como esquemas a encontrar



Sistema Hindley-Milner simple

- Las reglas se leen como esquemas a encontrar



- Dice que en una aplicación, la parte izquierda es una función y la parte derecha tiene el mismo tipo que el tipo del parámetro

Robin Milner

Arthur John Robin Gorell Milner
(13 de enero 1934 – 20 de marzo 2010) es un científico de la computación británico, fundador del Laboratorio de Fundamentos de Ciencias de la Computación (LFCS) de la Universidad de Edimburgo.
Desarrolló la *Lógica para Funciones Computables* (***LCF***) en 1972, una de las primeras herramientas para demostración automatizada de teoremas, el lenguaje ***ML*** (Meta-Language) en 1973, primer lenguaje con inferencia de tipos polimórfica (el sistema de tipos Hindley-Milner), pensado para implementar la LCF, y el *Cálculo de Sistemas Comunicantes* (***CCS***) en 1980 y su sucesor, el **π-cálculo** en 1992, para analizar sistemas concurrentes.

Sistema H-M: ejemplo 1

Sistema Hindley-Milner simple

- Ejemplo 1: `(\x->x+x) 2`

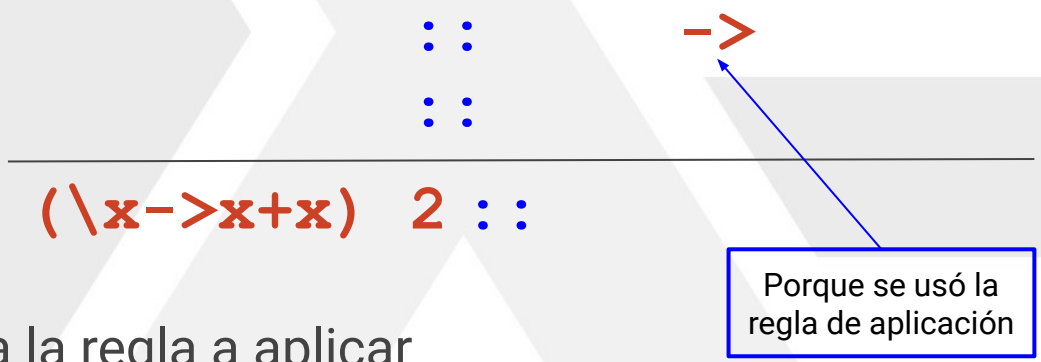
```
(\x->x+x) 2 ::
```

- Se plantea la pregunta

Sistema Hindley-Milner simple

$$\frac{f :: A \to B \qquad e :: A}{f\ e :: B}$$

- Ejemplo 1: `(\x->x+x) 2`

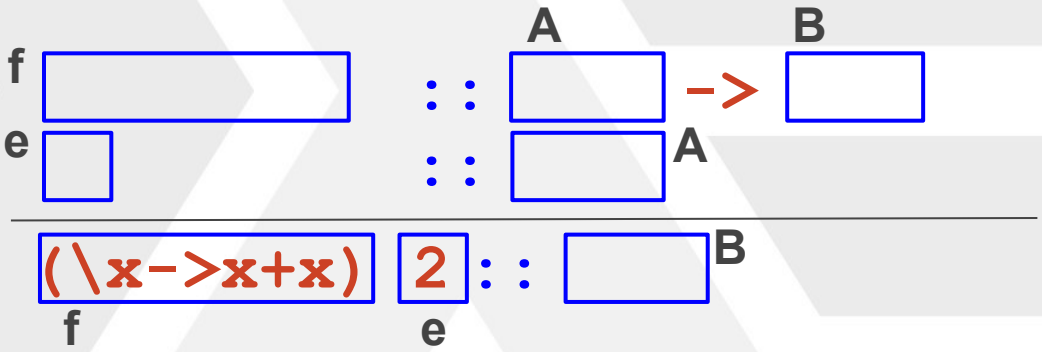


- Se determina la regla a aplicar

Sistema Hindley-Milner simple

f : : A -> B
e : : A
---
f e : : B

- Ejemplo 1: `(\x->x+x) 2`

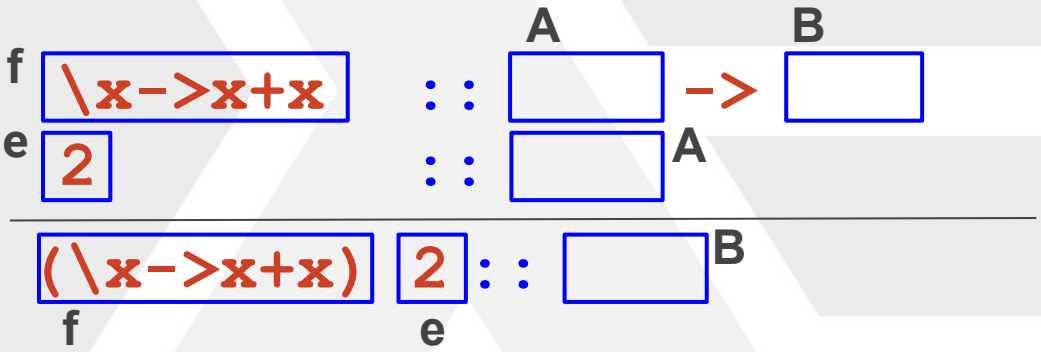


- Se analizan cuáles son los componentes

Sistema Hindley-Milner simple

```
f : : A -> B
e : : A
-------
f e : : B
```

- Ejemplo 1: `(\x->x+x) 2`



- Se comienzan a colocar las piezas en su lugar

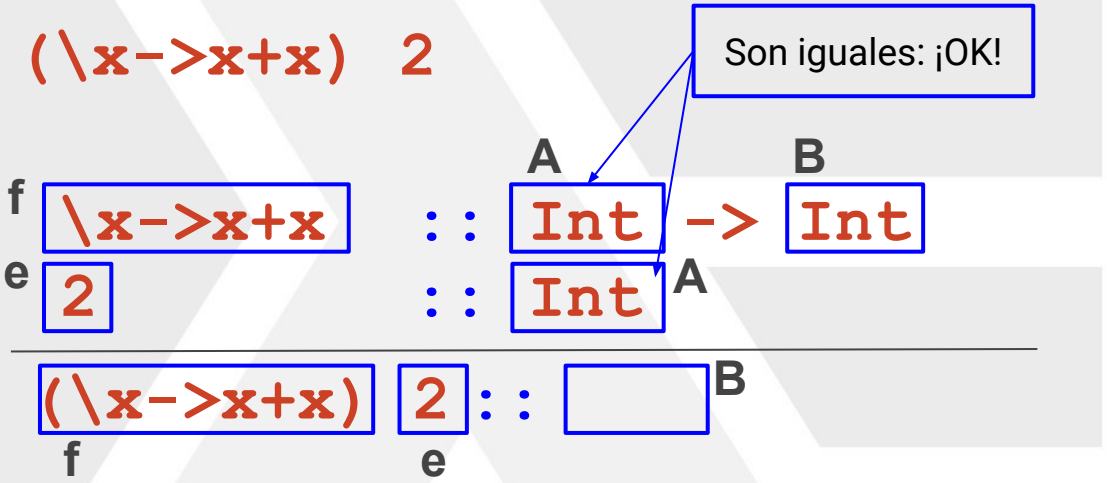
f : : A -> B
e : : A

---

f e : : B

Sistema Hindley-Milner simple

- Ejemplo 1: `(\x->x+x) 2`

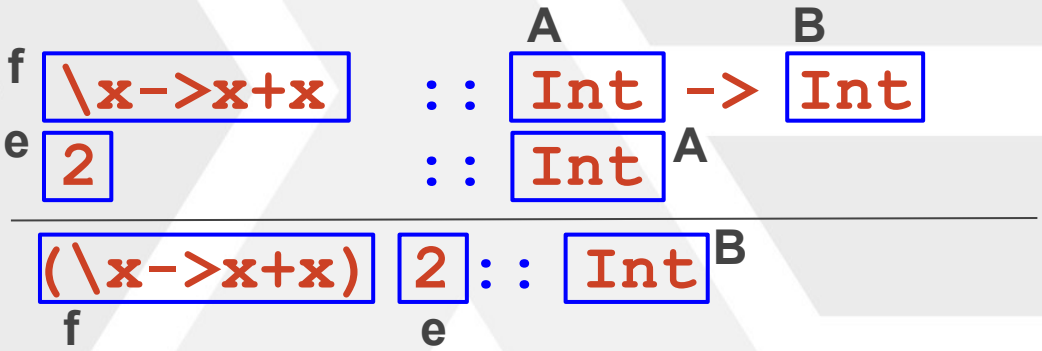


- Se resuelven los antecedentes y se verifican las condiciones

Sistema Hindley-Milner simple

$$\frac{f :: A \to B \qquad e :: A}{f\ e :: B}$$

- Ejemplo 1: `(\x->x+x) 2`



- Se concluye con el resultado: tiene tipo `Int`

```
(\x->x+x) 2 :: Int
```

Sistema H-M: ejemplo 2

Sistema Hindley-Milner simple

- Ejemplo 2: `2 + False`

```
2 + False ::
```

- Se plantea la pregunta

Sistema Hindley-Milner simple

$$\frac{e :: Int \qquad e' :: Int}{e + e' :: Int}$$

- Ejemplo 2: `2 + False`

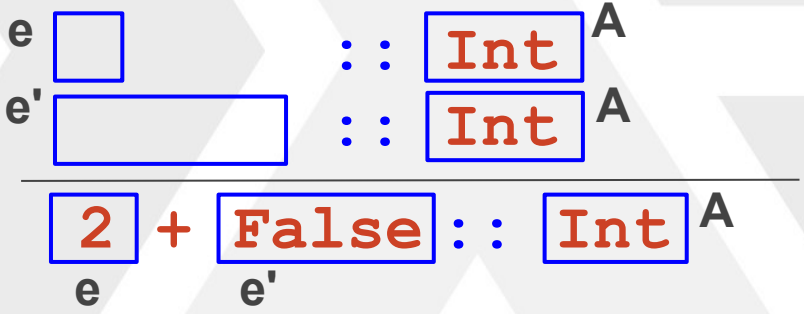
$$\frac{2 :: Int \qquad False :: Int}{2 + False :: Int}$$

- Se elige la regla a aplicar

Sistema Hindley-Milner simple

$$\frac{e :: Int \qquad e' :: Int}{e + e' :: Int}$$

- Ejemplo 2: `2 + False`



- Se analizan cuáles son los componentes

Sistema Hindley-Milner simple

$$\frac{e :: Int \qquad e' :: Int}{e + e' :: Int}$$

- Ejemplo 2: `2 + False`

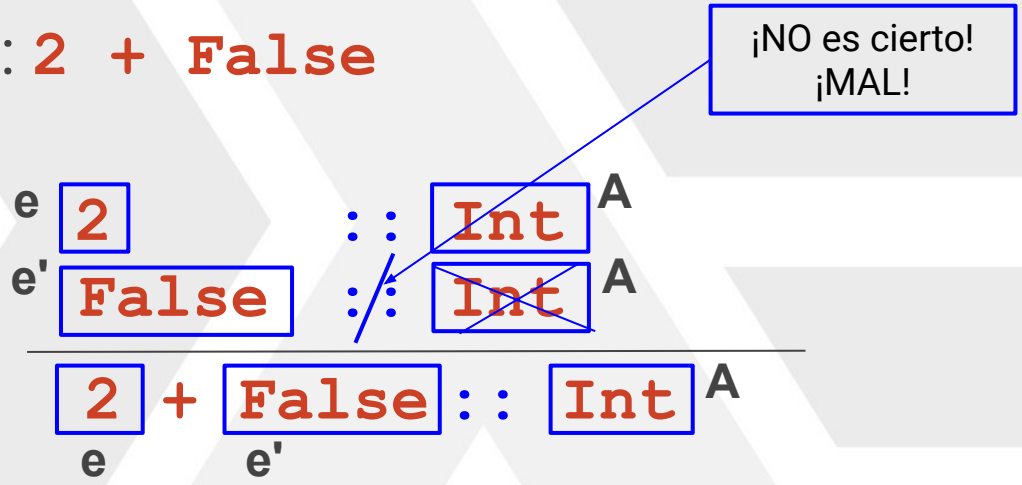
$$\frac{\begin{array}{ll} e & \texttt{2} :: \texttt{Int} \;\; A \\ e' & \texttt{False} :: \texttt{Int} \;\; A \end{array}}{\underset{e}{\texttt{2}} + \underset{e'}{\texttt{False}} :: \texttt{Int} \;\; A}$$

- Se comienzan a colocar las piezas en su lugar

Sistema Hindley-Milner simple

$$\frac{e :: \texttt{Int} \qquad e' :: \texttt{Int}}{e + e' :: \texttt{Int}}$$

- Ejemplo 2: `2 + False`

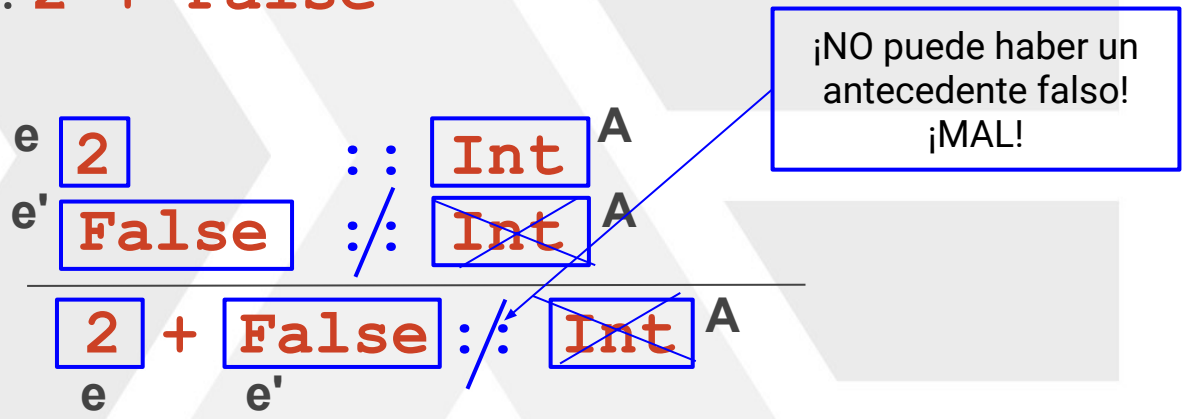


- Se verifican las condiciones de los antecedentes

Sistema Hindley-Milner simple

$$\frac{e :: Int \qquad e' :: Int}{e + e' :: Int}$$

- Ejemplo 2: `2 + False`



- Se concluye con el resultado: no es una expresión válida

`2 + False` :/:

Sistema H-M: ejemplo 3

Sistema Hindley-Milner simple

- Ejemplo 3: tipar definiciones `doble x = x+x`

```
doble ::
```

- Se plantea la pregunta

Sistema Hindley-Milner simple

- Ejemplo 3: tipar definiciones `doble x = x+x`

```
doble ::

doble x ::                    x + x ::
```

- Se desarrollan las partes de la definición

Sistema Hindley-Milner simple

$$\frac{f :: A \to B \qquad e :: A}{f\ e :: B}$$

$$\frac{e :: Int \qquad e' :: Int}{e + e' :: Int}$$

- Ejemplo 3: tipar definiciones `doble x = x+x`

$$\frac{doble :: \to \qquad :: }{doble\ x ::} \qquad \frac{:: Int \qquad :: Int}{x + x :: Int}$$

Porque se usó la regla de aplicación

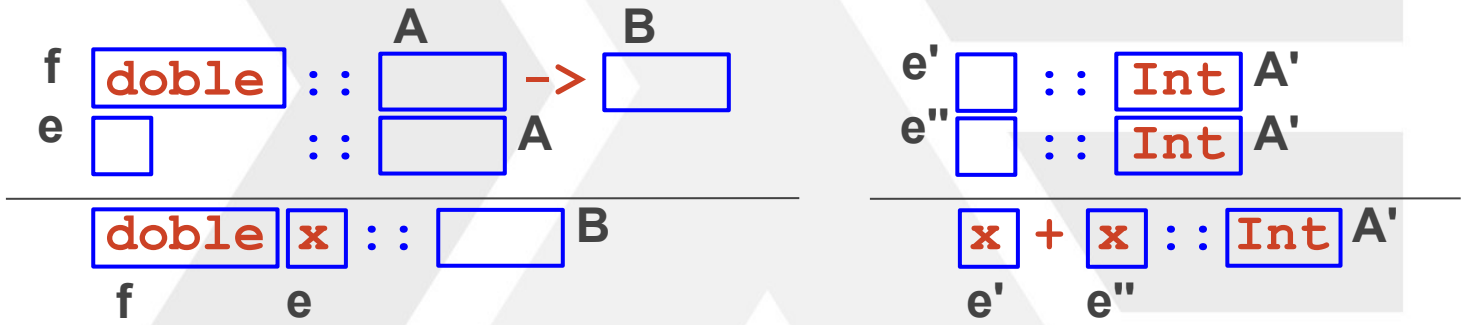
Porque se usó la regla de la suma

- Se determinan las reglas a aplicar

Sistema Hindley-Milner simple

$$\frac{f :: A \to B \quad e :: A}{f\ e :: B} \qquad \frac{e :: Int \quad e' :: Int}{e + e' :: Int}$$

- Ejemplo 3: tipar definiciones `doble x = x+x`

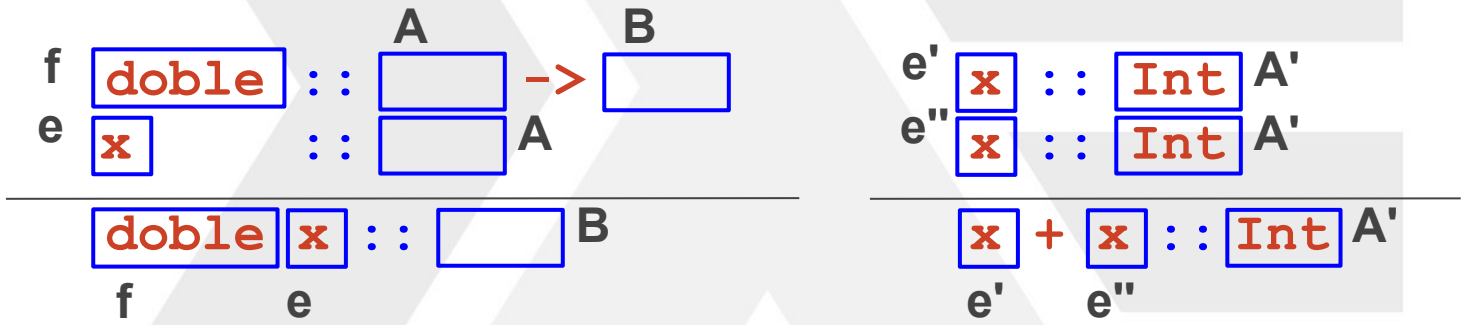


- Se analizan cuáles son los componentes

Sistema Hindley-Milner simple

$$\frac{f :: A \to B \quad e :: A}{f\ e :: B} \qquad \frac{e :: Int \quad e' :: Int}{e + e' :: Int}$$

- Ejemplo 3: tipar definiciones `doble x = x+x`

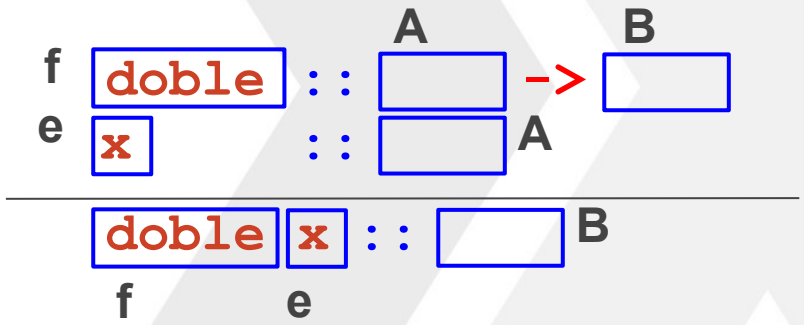


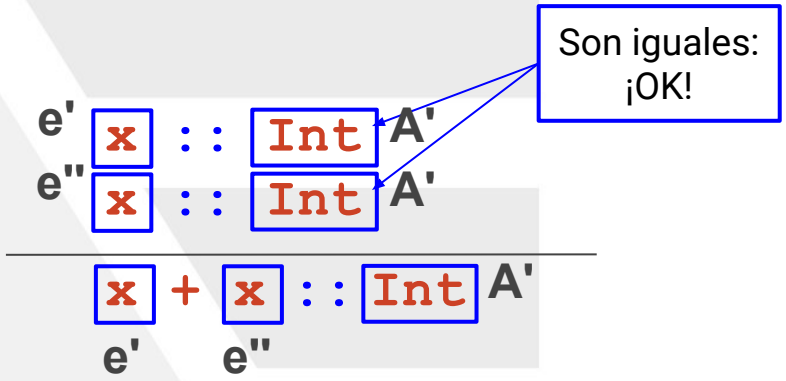
- Se comienzan a colocar las piezas en su lugar

Sistema Hindley-Milner simple

$$\frac{f :: A \to B \qquad e :: A}{f\ e :: B} \qquad \frac{e :: Int \qquad e' :: Int}{e + e' :: Int}$$

- Ejemplo 3: tipar definiciones `doble x = x+x`





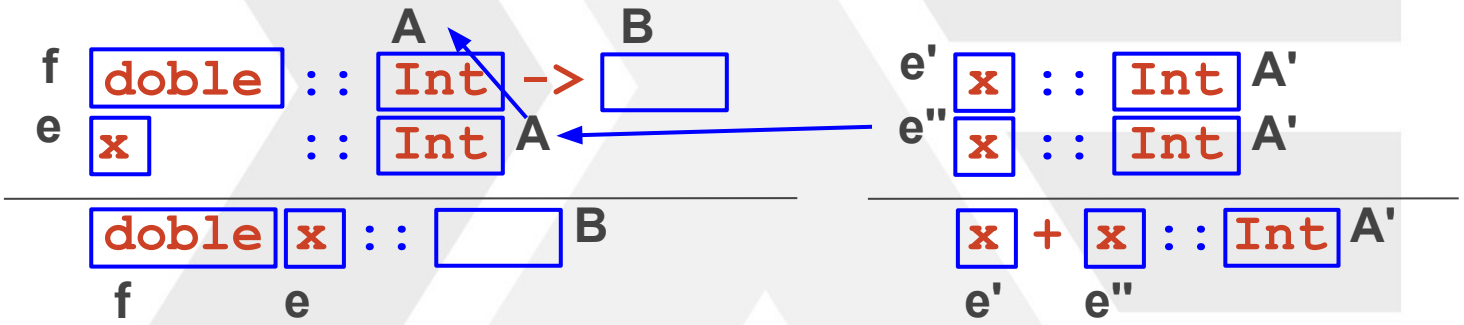
- Se verifican las condiciones

Sistema Hindley-Milner simple

$$\frac{f :: A \to B \quad e :: A}{f\ e :: B}$$

$$\frac{e :: Int \quad e' :: Int}{e + e' :: Int}$$

- Ejemplo 3: tipar definiciones `doble x = x+x`

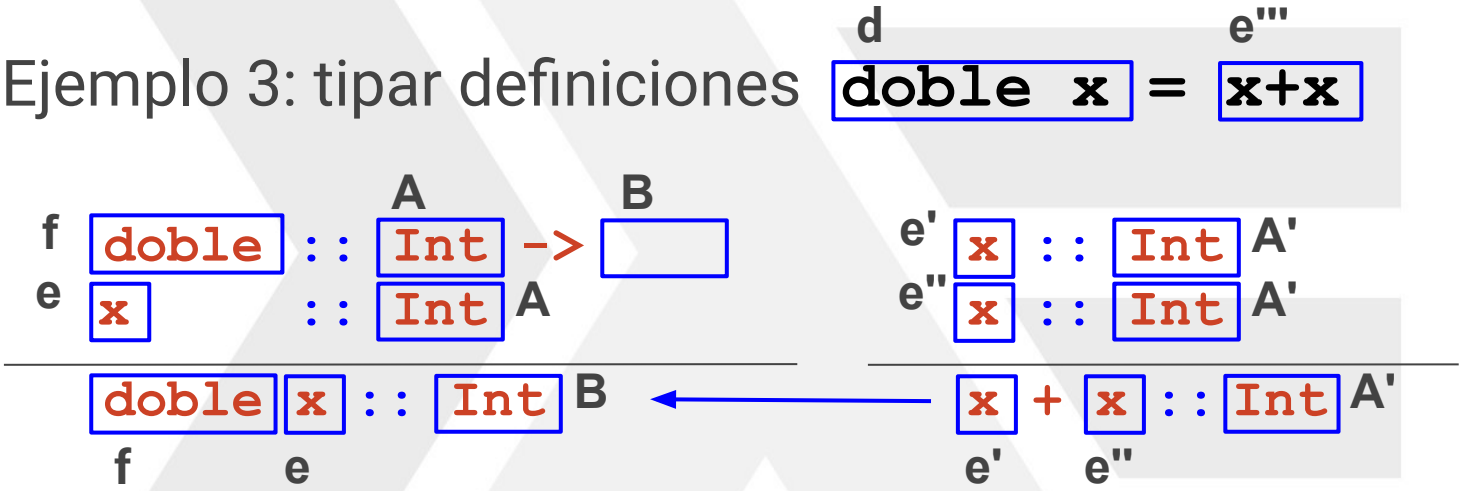


- Se propaga la información (es la misma `x` en todos lados)

Sistema Hindley-Milner simple

Si **d = e**
y **e :: A**
entonces **d :: A**

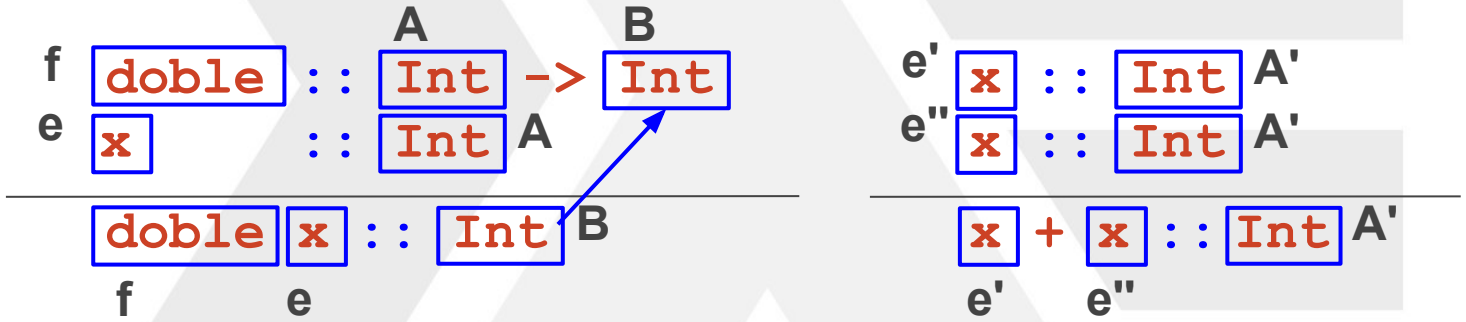
- Ejemplo 3: tipar definiciones



- Se propaga la información (con la regla de tipar definiciones)

Sistema Hindley-Milner simple

- Ejemplo 3: tipar definiciones **`doble x = x+x`**



- Se concluye con el resultado: función de números en números

```
doble :: Int -> Int
```

Sistema Hindley-Milner simple

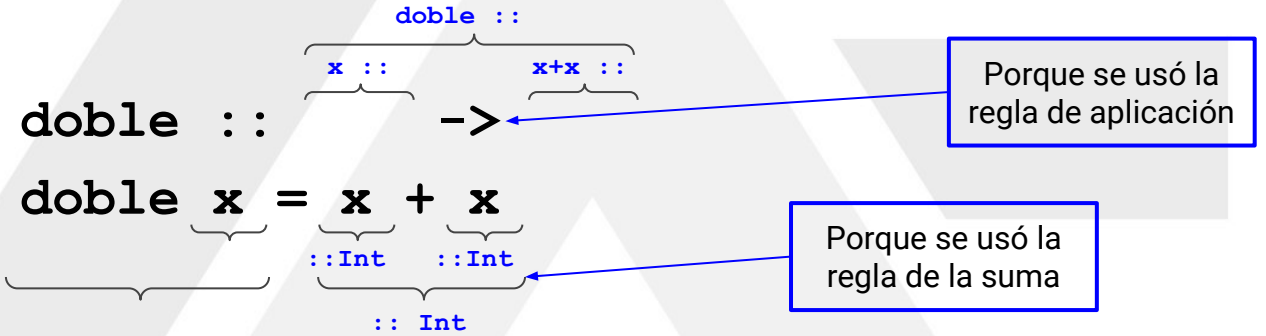
- Ejemplo 3: tipar definiciones `doble x = x+x`
- Se puede lograr lo mismo sin expandir las reglas

```
doble ::
doble x = x + x
```

- Se plantea la pregunta

Sistema Hindley-Milner simple

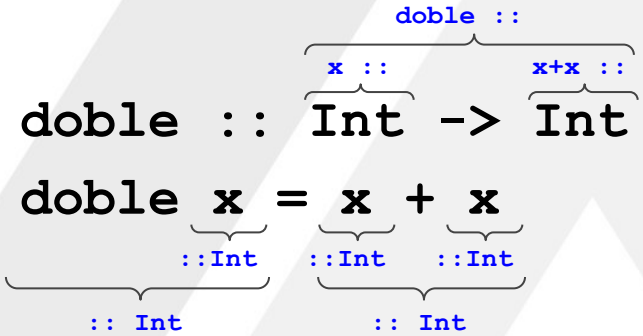
- Ejemplo 3: tipar definiciones **`doble x = x+x`**
- Se puede lograr lo mismo sin expandir las reglas



- Se analizan las partes y se determinan las reglas a aplicar

Sistema Hindley-Milner simple

- Ejemplo 3: tipar definiciones **`doble x = x+x`**
- Se puede lograr lo mismo sin expandir las reglas



- Se propaga la información

Sistema Hindley-Milner simple

- Ejemplo 3: tipar definiciones **`doble x = x+x`**
- Se puede lograr lo mismo sin expandir las reglas

```
doble :: Int -> Int
doble x = x + x
```

- Se concluye con el resultado

Sistema H-M: ejemplo 4

Sistema Hindley-Milner simple

- Ejemplo 4: `twice f = g where g x = f (f x)`

```
twice ::
```

- Se plantea la pregunta

Sistema Hindley-Milner simple

- Ejemplo 4: `twice f = g where g x = f (f x)`

```
twice ::

twice f ::          g ::

                    g x ::          f (f x) ::

                                                f x ::
```

- Se desarrollan las partes de la definición

Sistema Hindley-Milner simple

$$\frac{f :: A \to B \qquad e :: A}{f\ e :: B}$$

- Ejemplo 4: **`twice f = g where g x = f (f x)`**

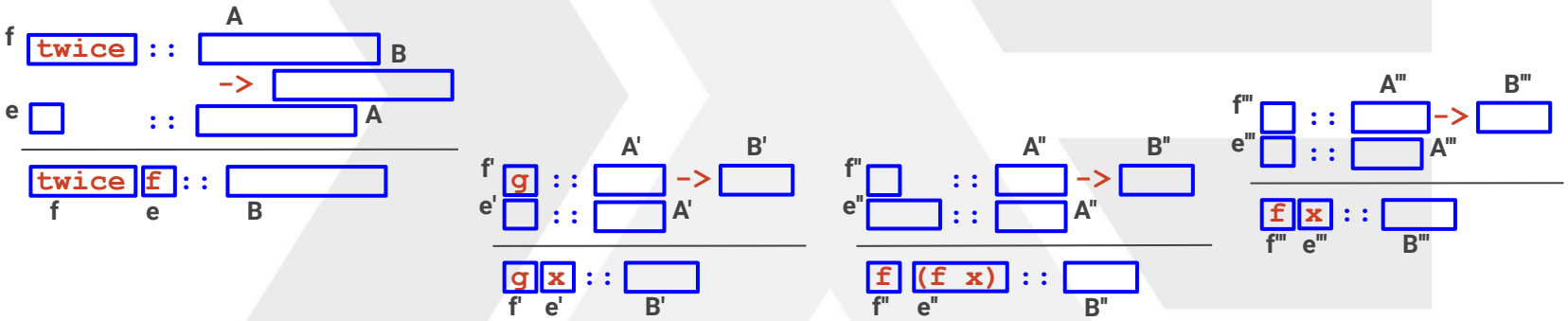
$$\frac{\texttt{twice} :: \quad \to \quad :: \qquad \texttt{twice f} :: \qquad \texttt{g} :: \quad \to \quad :: \qquad \texttt{g x} ::}{\texttt{twice} :: \quad \to \quad :: \qquad \texttt{twice f} :: \qquad \texttt{g} :: \quad \to \quad :: \qquad \texttt{g x} ::} \qquad \frac{:: \quad \to \quad :: \qquad \texttt{f (f x)} :: \qquad :: \quad \to \quad :: \qquad \texttt{f x} ::}{:: \quad \to \quad :: \qquad \texttt{f (f x)} :: \qquad :: \quad \to \quad :: \qquad \texttt{f x} ::}$$

- Se determinan las reglas a aplicar

Sistema Hindley-Milner simple

```
f    :: A -> B
e    :: A
-------------
f e  :: B
```

- Ejemplo 4: **`twice f = g where g x = f (f x)`**

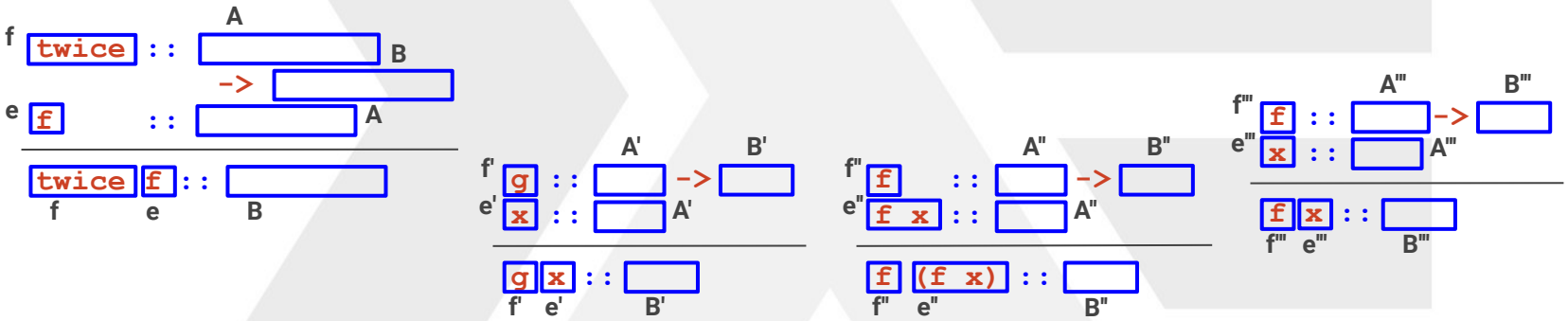


- Se analizan cuáles son los componentes

Sistema Hindley-Milner simple

```
f     :: A -> B
e     :: A
-------------
f e   :: B
```

- Ejemplo 4: **`twice f = g where g x = f (f x)`**

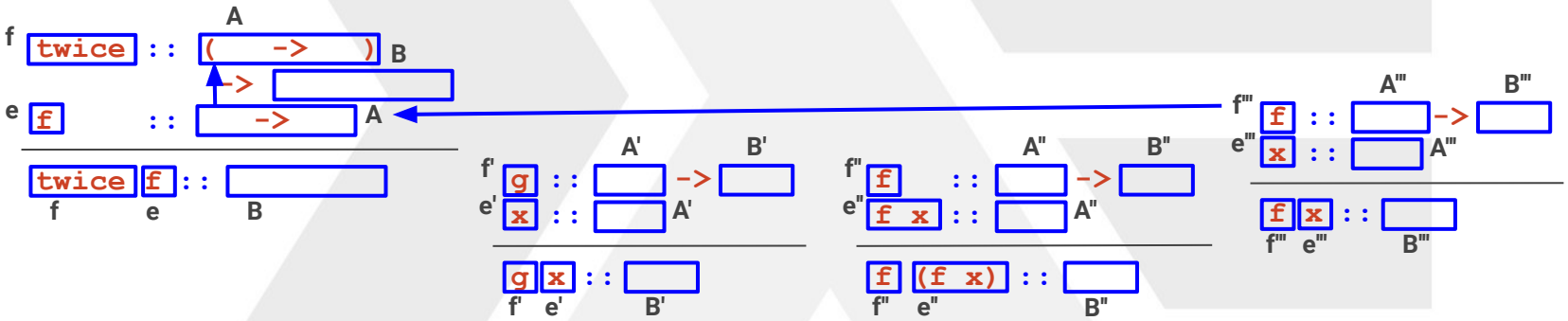


- Se comienzan a colocar las piezas en su lugar

Sistema Hindley-Milner simple

```
f     :: A -> B
e     :: A
-----------------
f e   :: B
```

- Ejemplo 4: `twice f = g where g x = f (f x)`

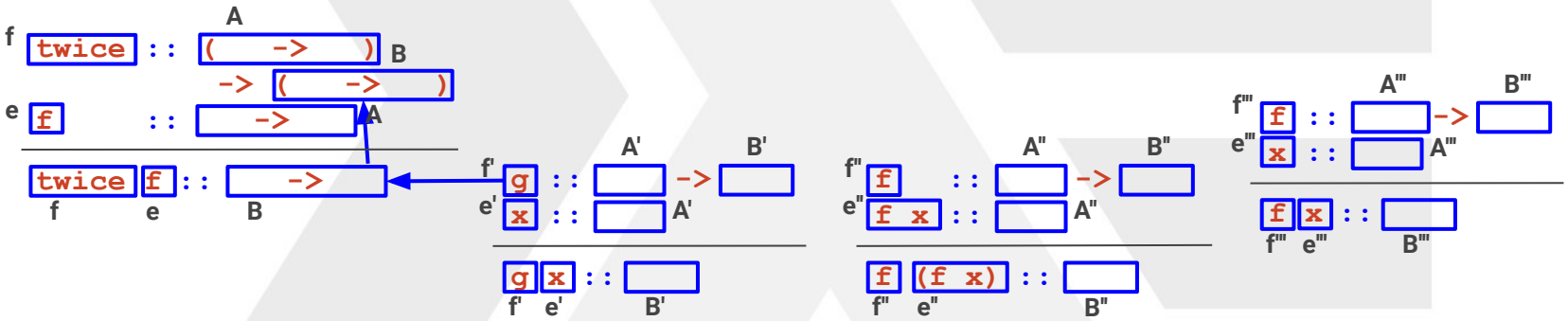


- Se propaga la información (es la misma `f` en todos lados)

Sistema Hindley-Milner simple

```
f    :: A -> B
e    :: A
-------------
f e  :: B
```

- Ejemplo 4: **`twice f = g where g x = f (f x)`**

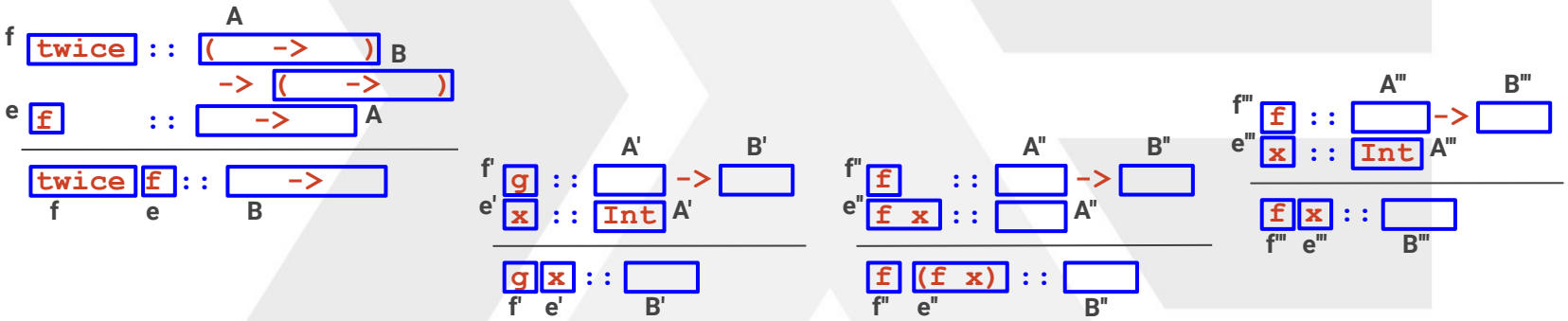


- Se propaga la información (con la regla de tipar definiciones)

Sistema Hindley-Milner simple

```
f    :: A -> B
e    :: A
-----------
f e  :: B
```

- Ejemplo 4: **`twice f = g where g x = f (f x)`**

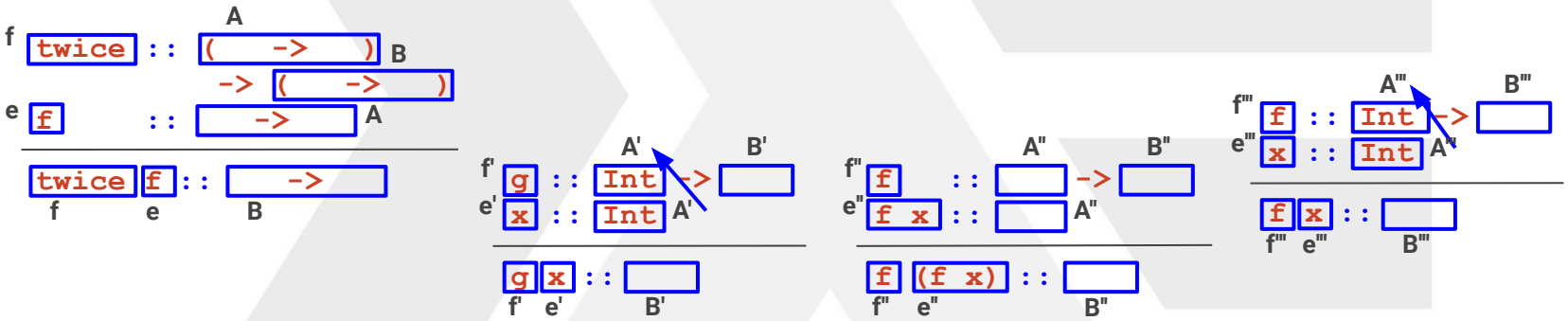


- Se decide el tipo de **`x`**: por ejemplo, **`Int`** (porque no hay restricciones)

Sistema Hindley-Milner simple

```
f    :: A -> B
e    :: A
-------------
f e  :: B
```

- Ejemplo 4: **`twice f = g where g x = f (f x)`**

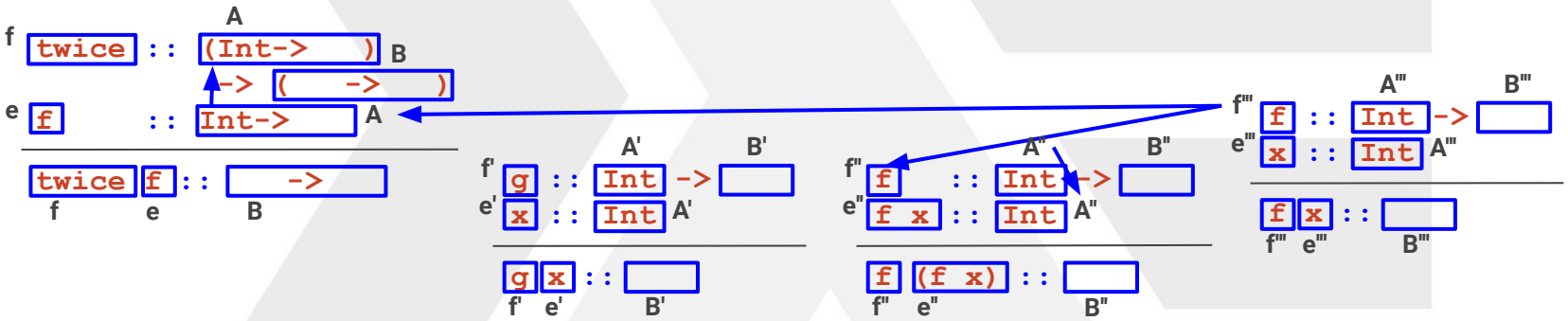


- Se propaga la información (por igualdad de componentes)

Sistema Hindley-Milner simple

```
f     :: A -> B
e     :: A
-------------
f e   :: B
```

- Ejemplo 4: **`twice f = g where g x = f (f x)`**

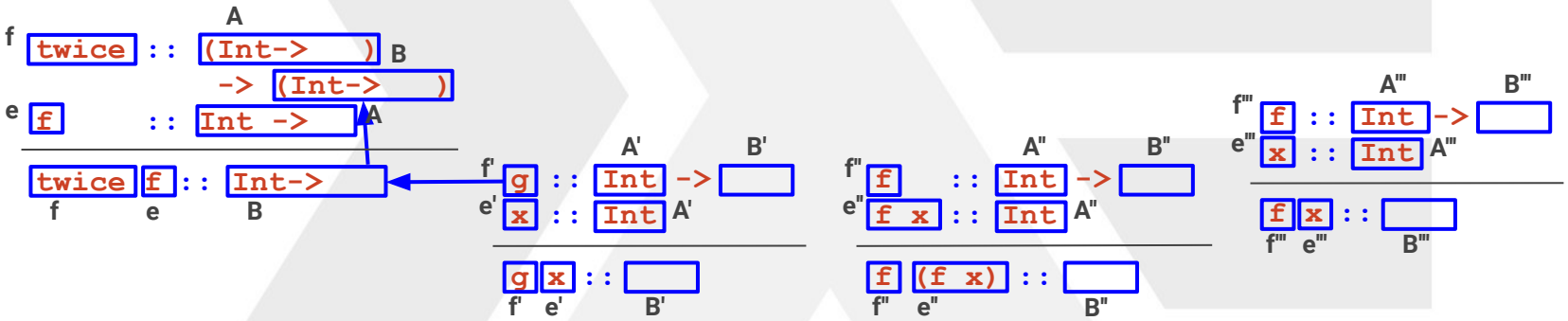


- Se propaga la información (es la misma **`f`** en todos lados)

Sistema Hindley-Milner simple

```
f    :: A -> B
e    :: A
-------------
f e  :: B
```

- Ejemplo 4: **`twice f = g where g x = f (f x)`**

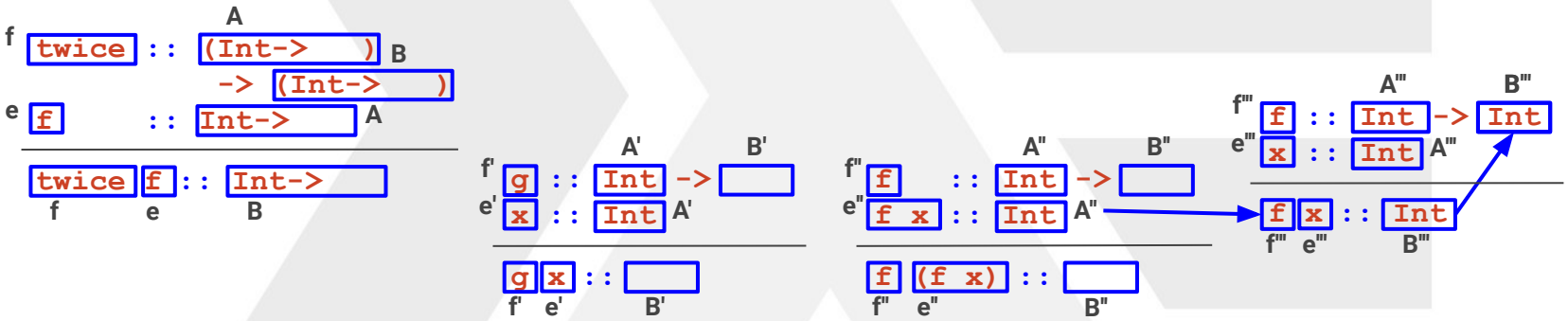


- Se propaga la información (con la regla de tipar definiciones)

Sistema Hindley-Milner simple

```
f    :: A -> B
e    :: A
-------------
f e  :: B
```

- Ejemplo 4: **`twice f = g where g x = f (f x)`**

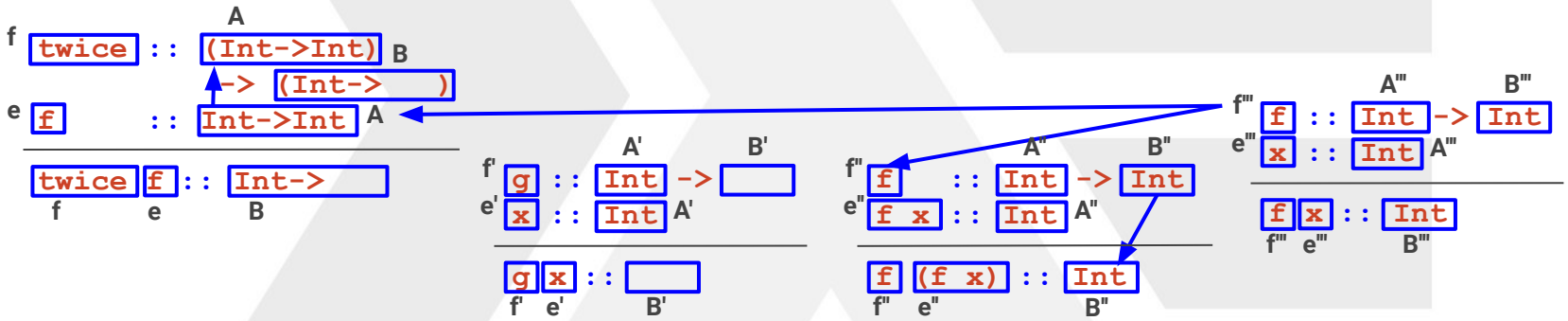


- Se propaga la información (**`f x`** es la misma ambas veces)

Sistema Hindley-Milner simple

```
f    :: A -> B
e    :: A
-------------
f e  :: B
```

- Ejemplo 4: **`twice f = g where g x = f (f x)`**

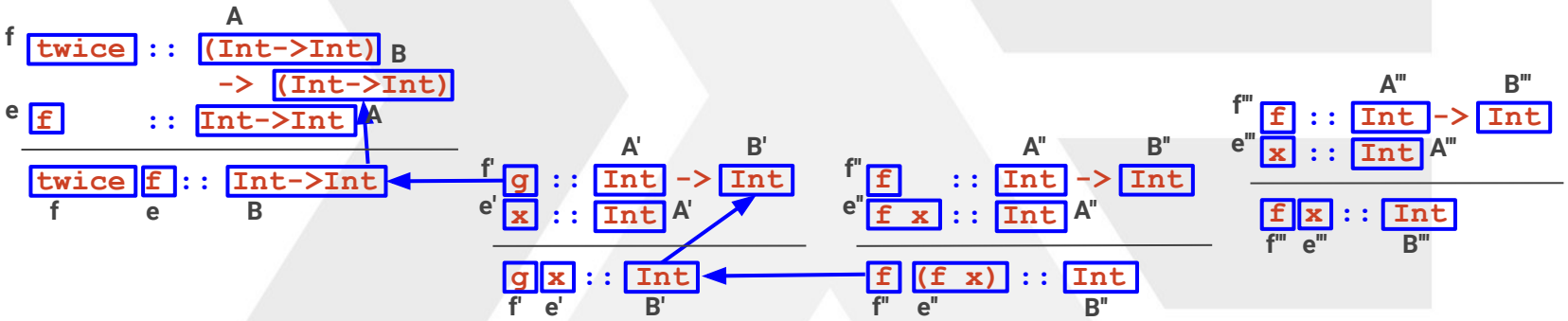


- Se propaga la información (es la misma **`f`** en todos lados)

Sistema Hindley-Milner simple

```
f    :: A -> B
e    :: A
-------------
f e  :: B
```

- Ejemplo 4: **`twice f = g where g x = f (f x)`**

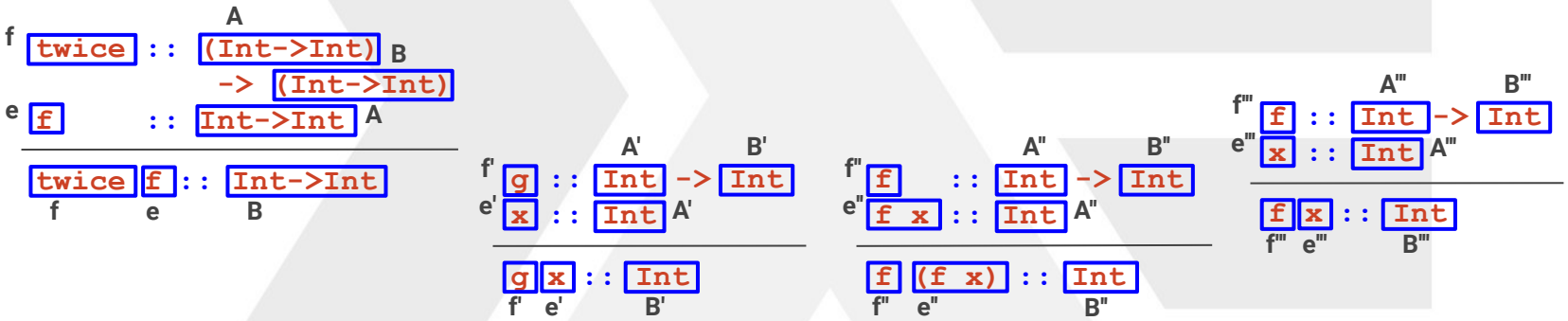


- Se propaga la información (con la regla de tipar definiciones)

Sistema Hindley-Milner simple

```
f    :: A -> B
e    :: A
-------------
f e  :: B
```

- Ejemplo 4: **`twice f = g where g x = f (f x)`**



- Se concluye con el resultado: función de números en números

`twice :: (Int -> Int) -> (Int -> Int)`

Sistema Hindley-Milner simple

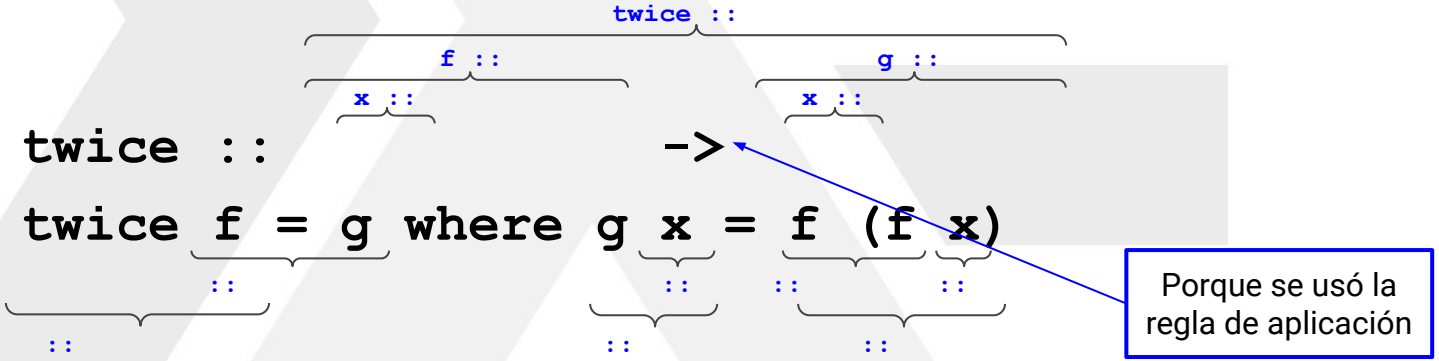
- Ejemplo 4: `twice f = g where g x = f (f x)`
- Se puede lograr lo mismo sin expandir las reglas

```
twice ::
twice f = g where g x = f (f x)
```

- Se plantea la pregunta

Sistema Hindley-Milner simple

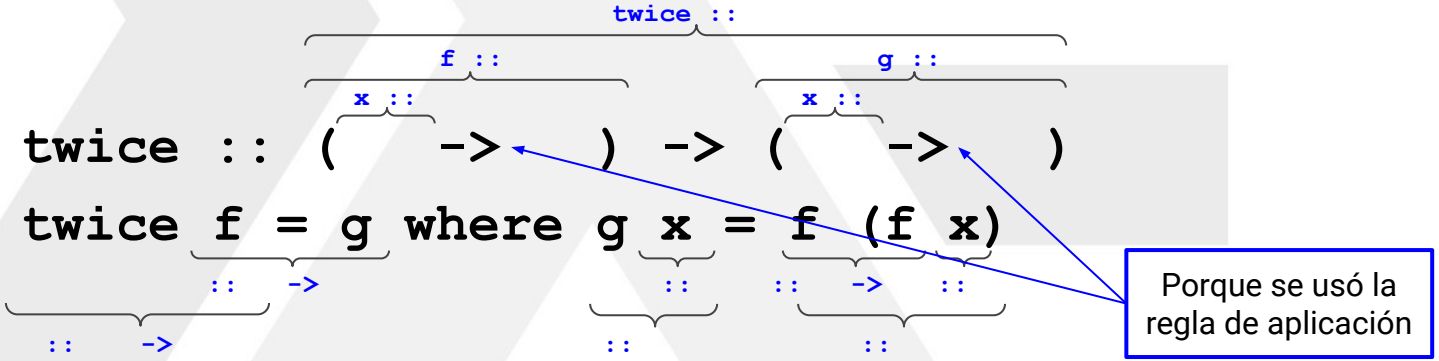
- Ejemplo 4: `twice f = g where g x = f (f x)`
- Se puede lograr lo mismo sin expandir las reglas



- Se analizan las partes y se determinan las reglas a aplicar

Sistema Hindley-Milner simple

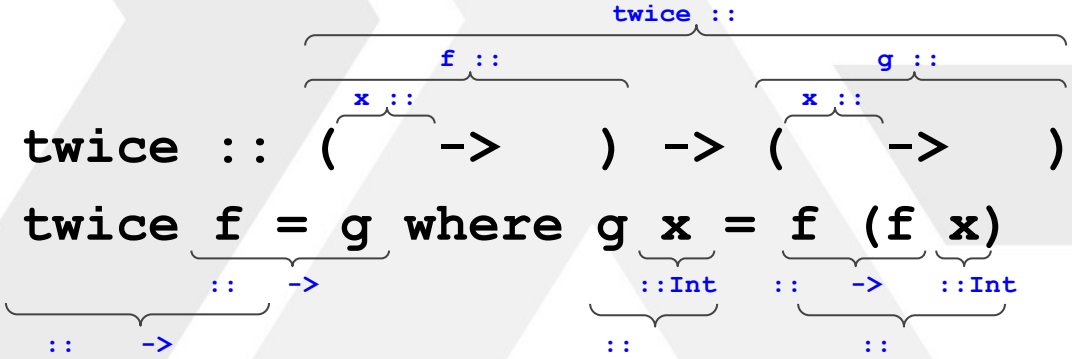
- Ejemplo 4: `twice f = g where g x = f (f x)`
- Se puede lograr lo mismo sin expandir las reglas



- Se deciden más reglas y se propaga la información

Sistema Hindley-Milner simple

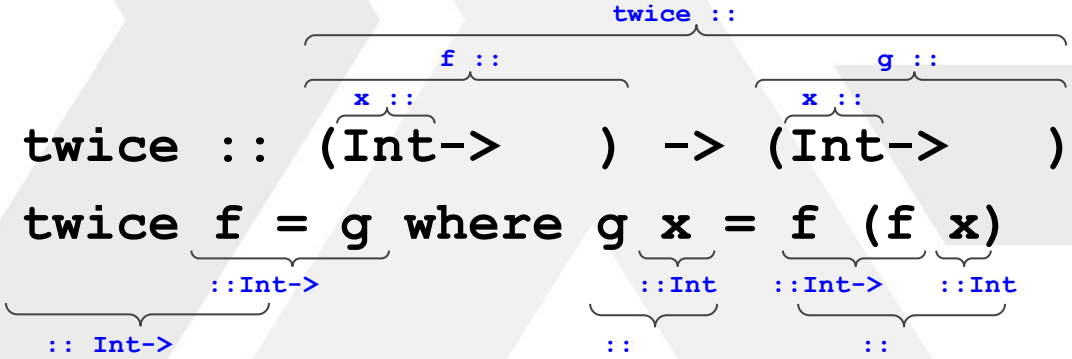
- Ejemplo 4: **`twice f = g where g x = f (f x)`**
- Se puede lograr lo mismo sin expandir las reglas



- Se elige un tipo para **`x`** (p.ej., **`Int`**, porque no hay restricciones)

Sistema Hindley-Milner simple

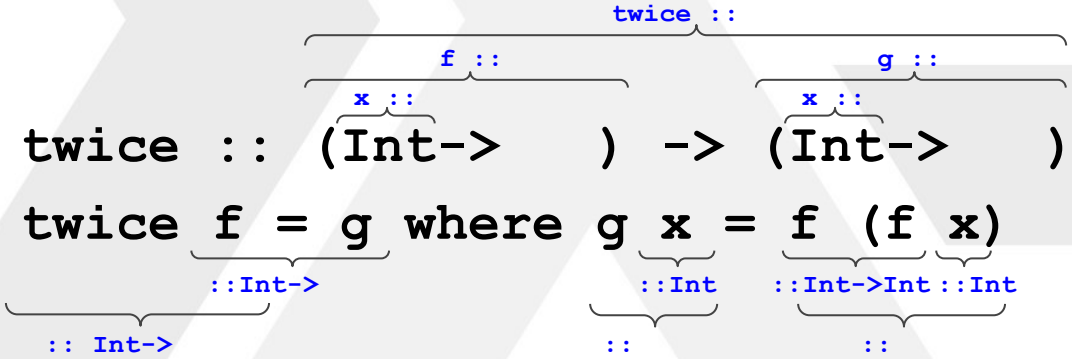
- Ejemplo 4: **`twice f = g where g x = f (f x)`**
- Se puede lograr lo mismo sin expandir las reglas



- Se propaga la información (a todos los lugares donde está **`x`**)

Sistema Hindley-Milner simple

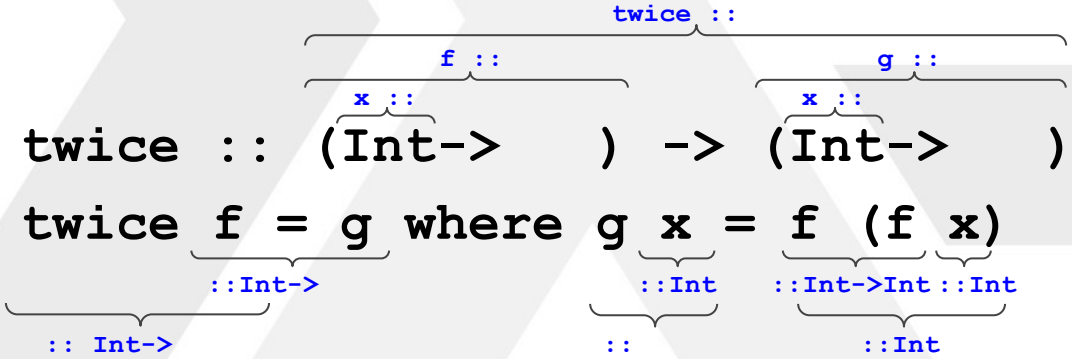
- Ejemplo 4: `twice f = g where g x = f (f x)`
- Se puede lograr lo mismo sin expandir las reglas



- Se propaga la información (porque `(f x)` es argumento de `f`)

Sistema Hindley-Milner simple

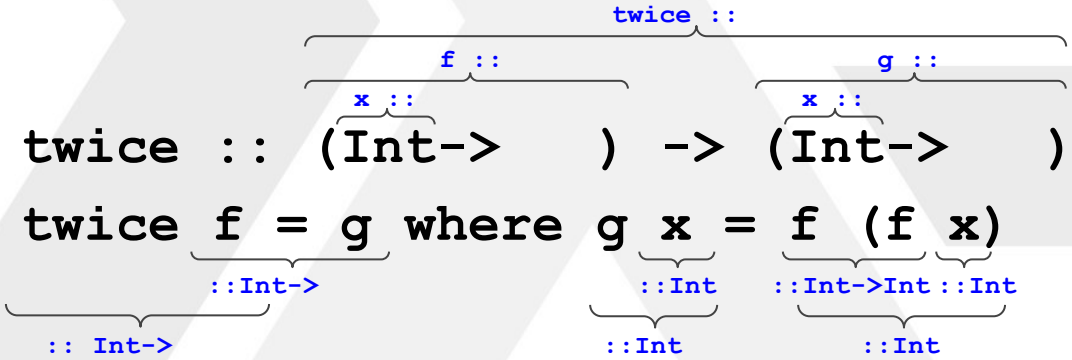
- Ejemplo 4: `twice f = g where g x = f (f x)`
- Se puede lograr lo mismo sin expandir las reglas



- Se propaga la información (por igualdad de componentes)

Sistema Hindley-Milner simple

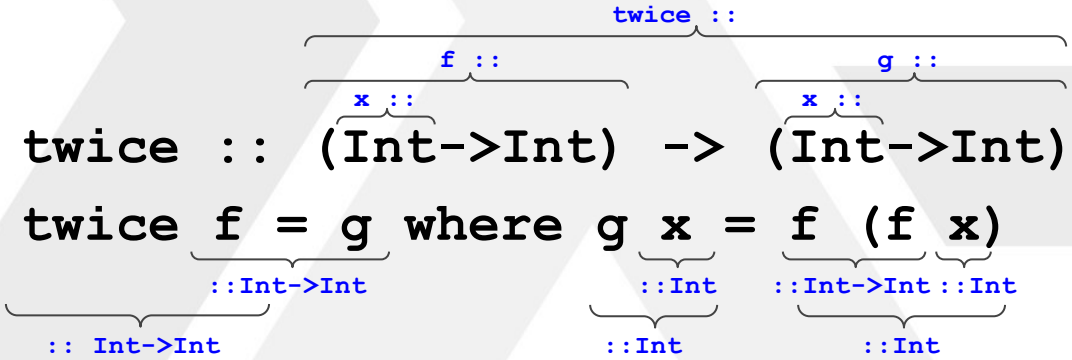
- Ejemplo 4: `twice f = g where g x = f (f x)`
- Se puede lograr lo mismo sin expandir las reglas



- Se propaga la información (con la regla de tipar definiciones)

Sistema Hindley-Milner simple

- Ejemplo 4: **`twice f = g where g x = f (f x)`**
- Se puede lograr lo mismo sin expandir las reglas



- Se propaga la información (por igualdad de componentes)

Sistema Hindley-Milner simple

- Ejemplo 4: `twice f = g where g x = f (f x)`
- Se puede lograr lo mismo sin expandir las reglas

```
twice :: (Int->Int) -> (Int->Int)
twice f = g where g x = f (f x)
```

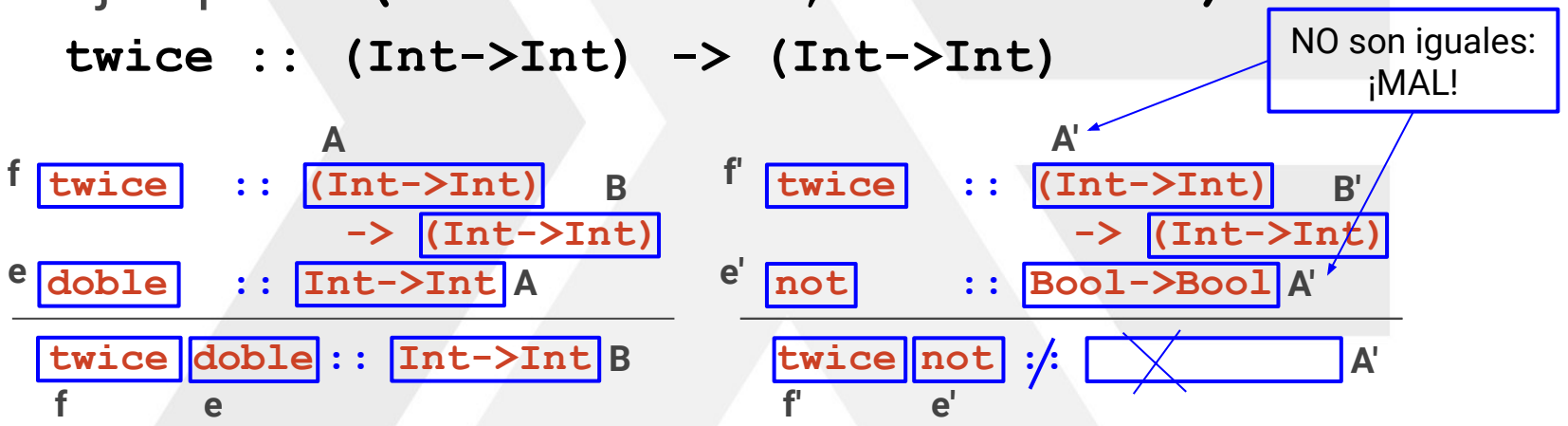
- Se concluye con la respuesta

Polimorfismo

Polimorfismo: motivación

- Ejemplo 5: **`(twice doble, twice not)`**

```
twice :: (Int->Int) -> (Int->Int)
```

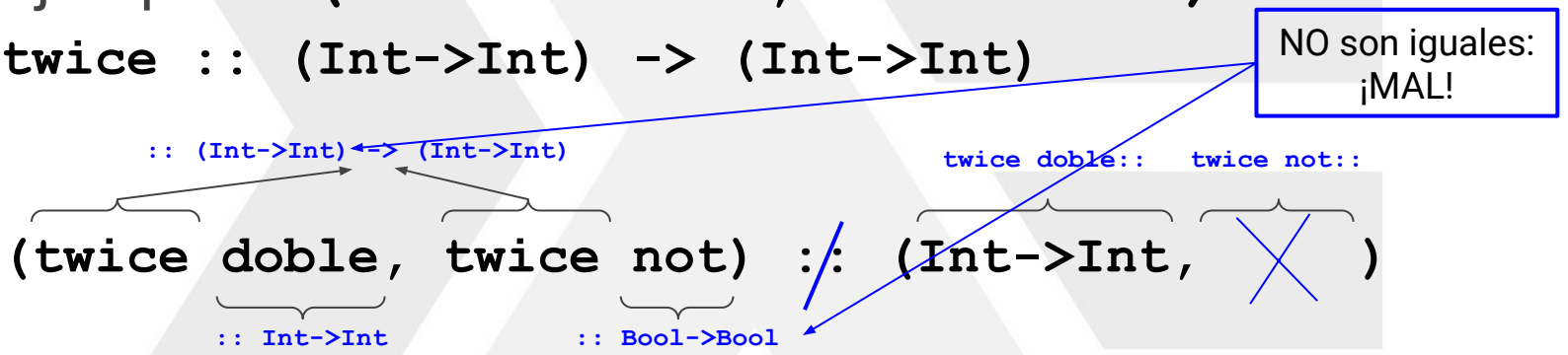


- ¿Por qué no se pueden darle tipo a ambas partes?

Polimorfismo: motivación

- Ejemplo 5: **`(twice doble, twice not)`**

```
twice :: (Int->Int) -> (Int->Int)
```



- ¿Por qué no se pueden darle tipo a ambas partes?

Polimorfismo: motivación

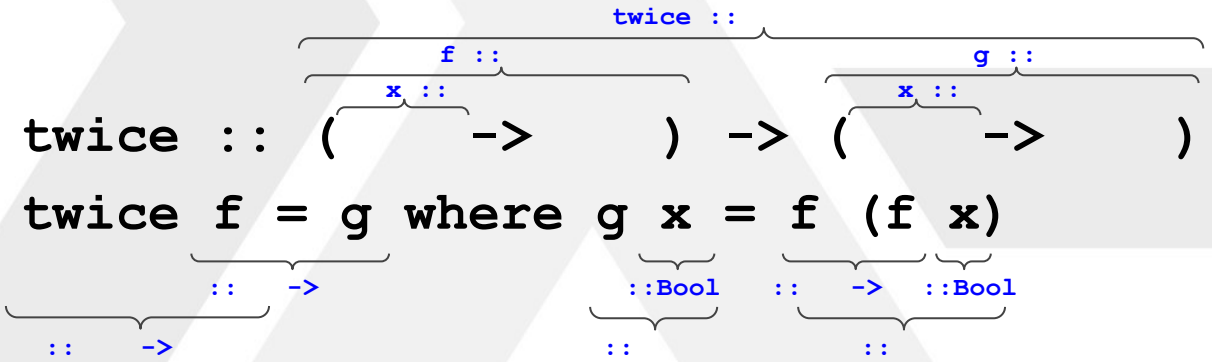
- Ejemplo 5: **`(twice doble, twice not)`**

  `twice :: (Int->Int) -> (Int->Int)`

- ¿Por qué no se pueden darle tipo a ambas partes?
  - En el ej.4 el tipo de **`x`** fue elegido como **`Int`**
  - Pero entonces **`not`** no puede ser argumento de **`twice`**
  - ¿Podría elegirse otra cosa?

Polimorfismo: motivación

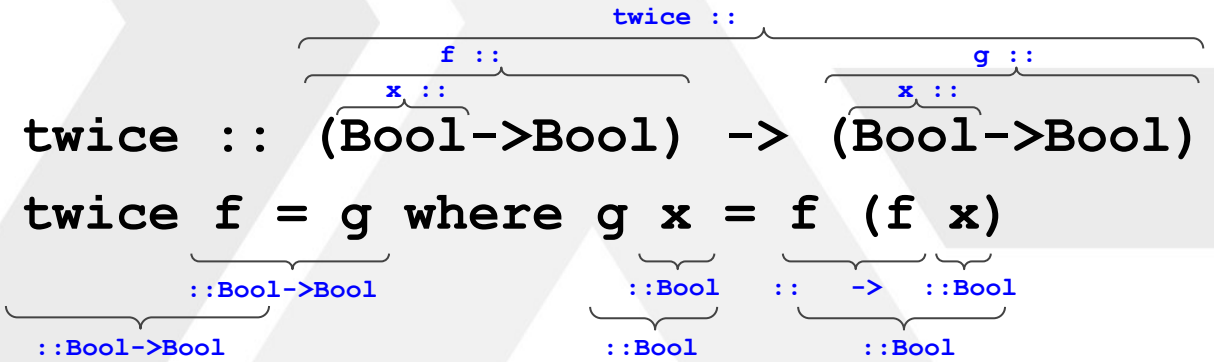
- Ejemplo 4, revisado: **`twice`**
  - (partiendo desde el punto de elección del tipo de **`x`**)



- Se elige ***otro*** tipo para **`x`** (p.ej., **`Bool`**, porque no hay restricciones)

Polimorfismo: motivación

- Ejemplo 4, revisado: **`twice`**
  - (partiendo desde el punto de elección del tipo de **`x`**)

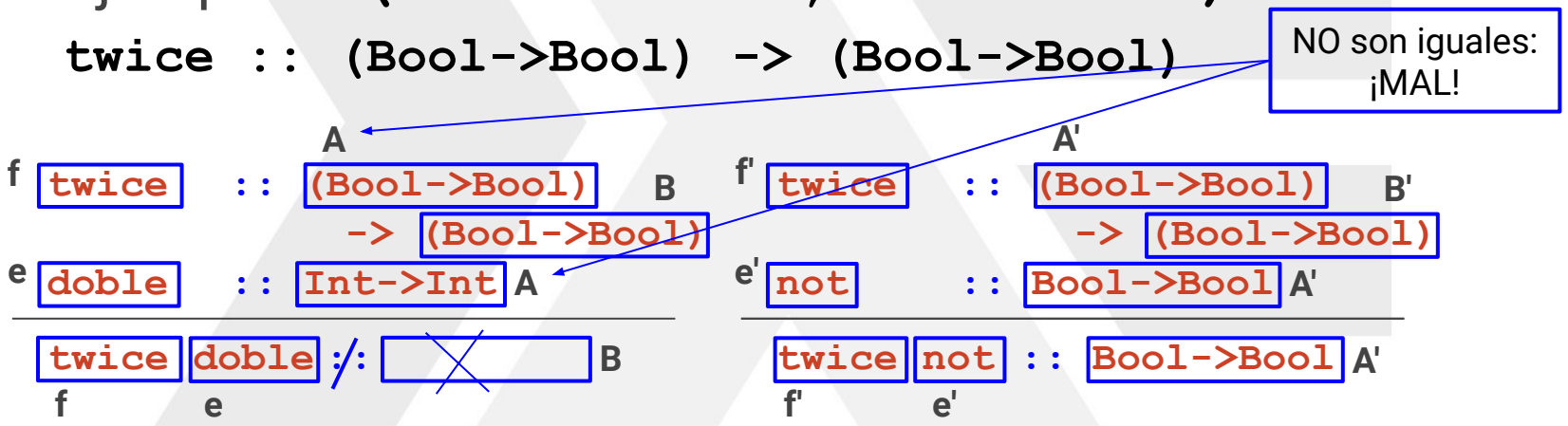


- Y se propaga la información

Polimorfismo: motivación

- Ejemplo 5: **`(twice doble, twice not)`**

  `twice :: (Bool->Bool) -> (Bool->Bool)`

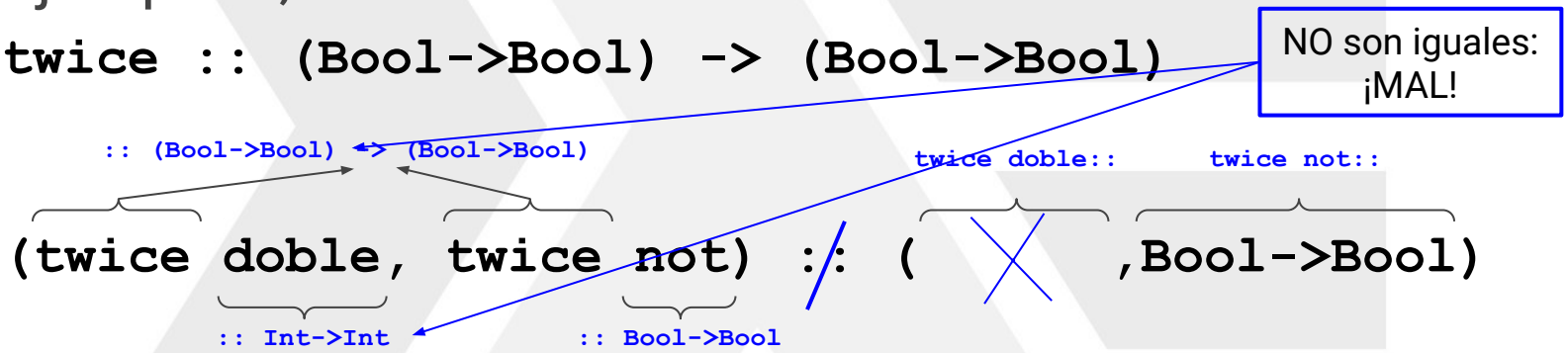


- ¿Por qué no se pueden darle tipo a ambas partes?

Polimorfismo: motivación

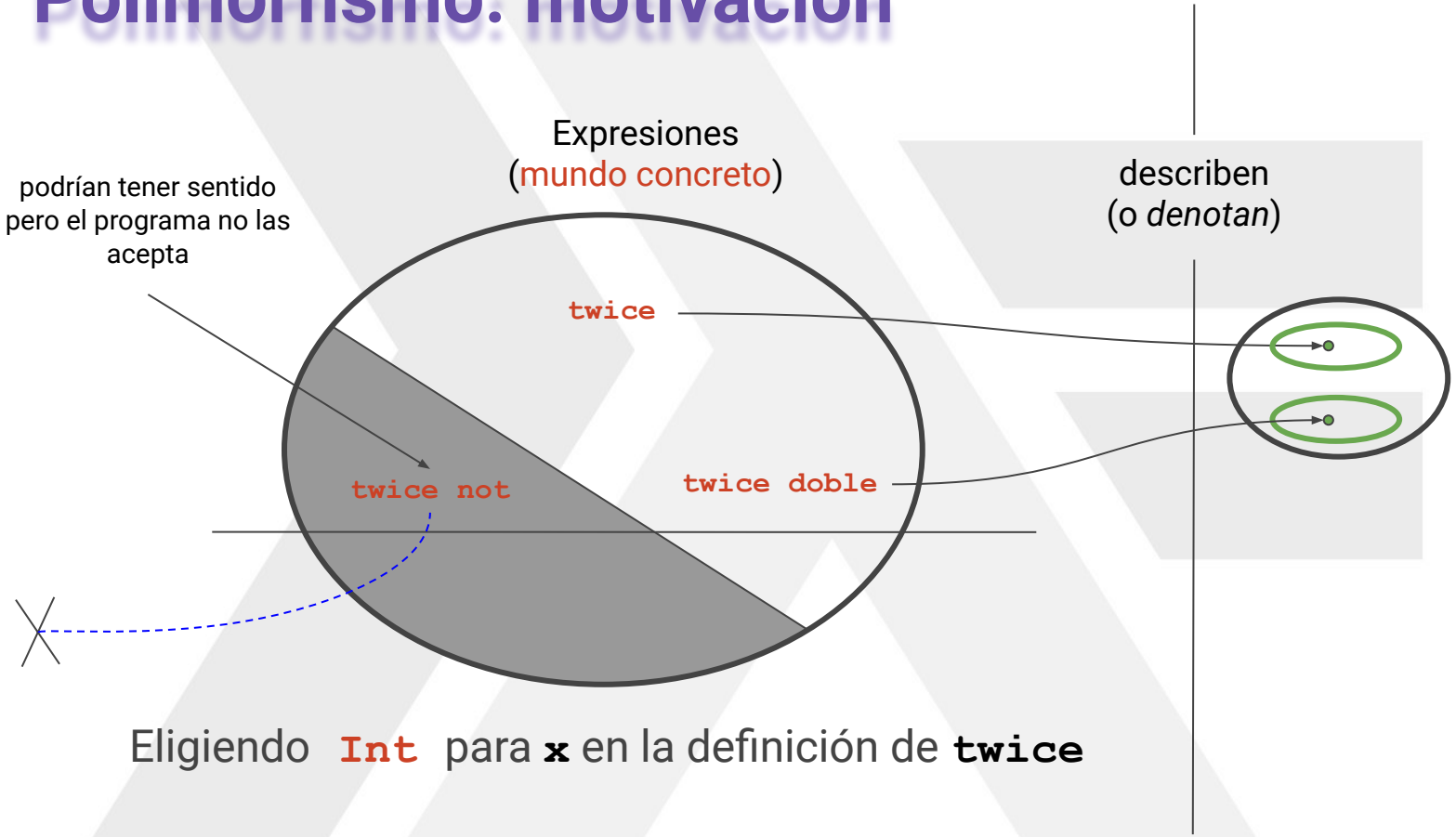
- Ejemplo 5, revisado

```
twice :: (Bool->Bool) -> (Bool->Bool)
```



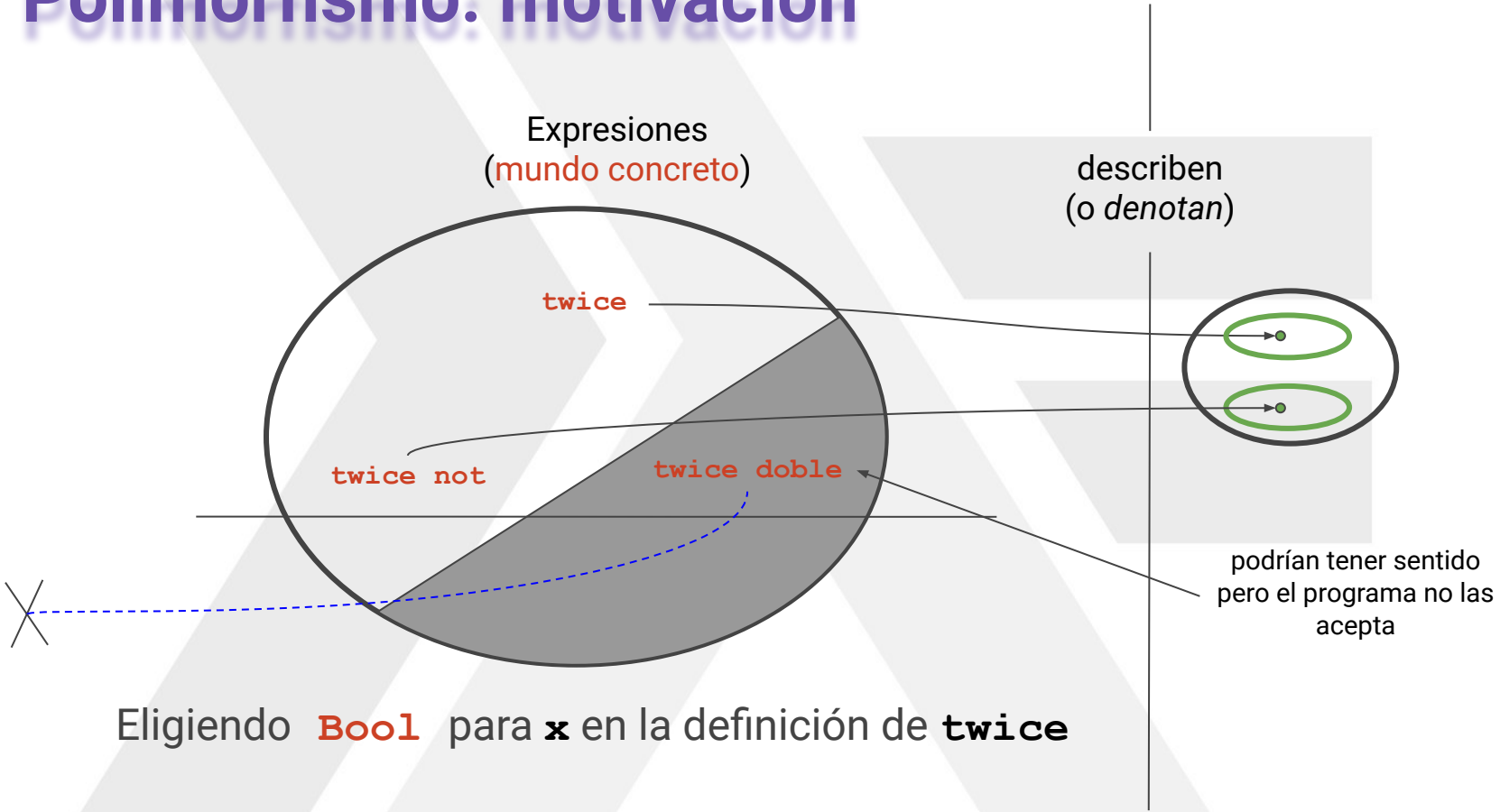
- ¿Y entonces? ¿Cuál debería ser la elección para **x**?

Polimorfismo: motivación



Eligiendo `Int` para **`x`** en la definición de **`twice`**

Polimorfismo: motivación



Eligiendo `Bool` para **`x`** en la definición de **`twice`**

Polimorfismo: motivación

- ¿Cuál debería ser el tipo de `twice`?

```
twice :: (Int->Int) -> (Int->Int)
twice :: (Bool->Bool) -> (Bool->Bool)
twice :: ( ?? -> ?? ) -> ( ?? -> ?? )
```

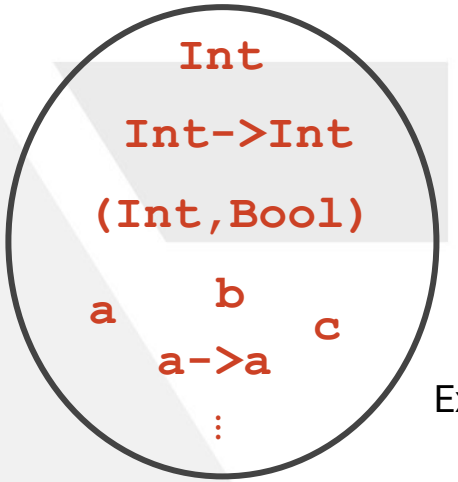
- Hay infinitas elecciones posibles para este **??**
- Al forzar la elección, se invalidan infinitas otras
- ¿Cómo hacer entonces para no forzar la elección, cuando hay que elegir un tipo?

Polimorfismo

- ¿Cómo hacer para no forzar la elección del tipo?
- Necesitamos un “tipo” que permita elegir después
- ¡Un mecanismo para esto es tener ***variables de tipo***!
- Se *agregan* variables al conjunto de expresiones de tipo
  - Se utilizarán letras minúsculas, `a`, `b`, `c`, ...
  - Así, `twice :: ( a -> a ) -> ( a -> a )`

Sistema Hindley-Milner polimórfico

- Se agregan variables a los tipos
  - Básicos
    - `Int`, `Bool`, etc.
  - Compuestos
    - **A** `->` **B**, `(`**A**`,`**B**`)`, etc.
  - Variables (para polimorfismo)
    - `a`, `b`, `c`, ...



Expresiones de tipo

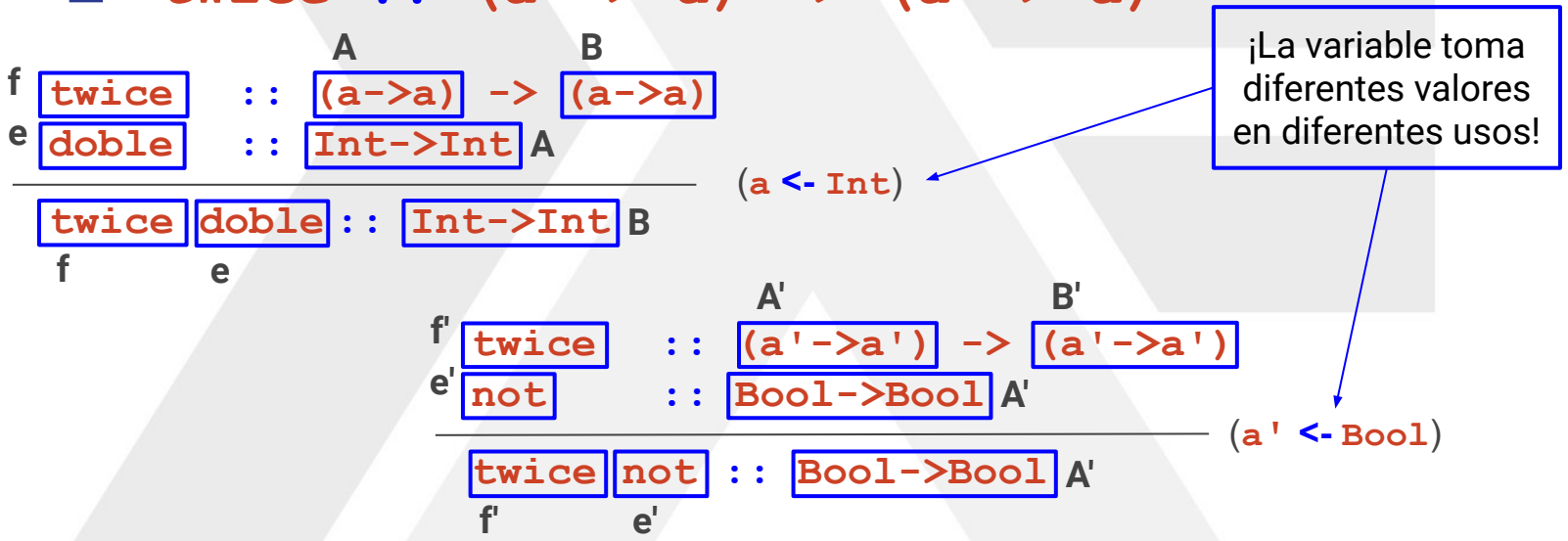
Sistema Hindley-Milner polimórfico

- ¿Qué significa el nuevo tipo de twice?
  - `twice :: (a -> a) -> (a -> a)`
  - Se lee: "*`twice` es una función que dada una función que transforma elementos de **algún tipo** `a` en otros elementos de ese tipo, retorna otra función de ese mismo tipo*"
  - El tipo `a` puede ser ***instanciado*** a diferentes tipos, en diferentes usos

```
twice f = g
  where g x = f (f x)
```

Sistema Hindley-Milner polimórfico

- Instanciación de **a** según el contexto
  - `twice :: (a -> a) -> (a -> a)`



Sistema Hindley-Milner polimórfico

- Definición de ***polimorfismo paramétrico***
  - Un sistema de tipos posee *polimorfismo paramétrico* cuando, para expresiones que admiten infinitos tipos, puede asignar uno más general que pueda transformarse en cualquiera de ellos
  - Es una característica del sistema de tipos
  - Se dice “paramétrico” porque `a` funciona como un parámetro

Sistema H-M: ejemplo 5

Sistema Hindley-Milner polimórfico

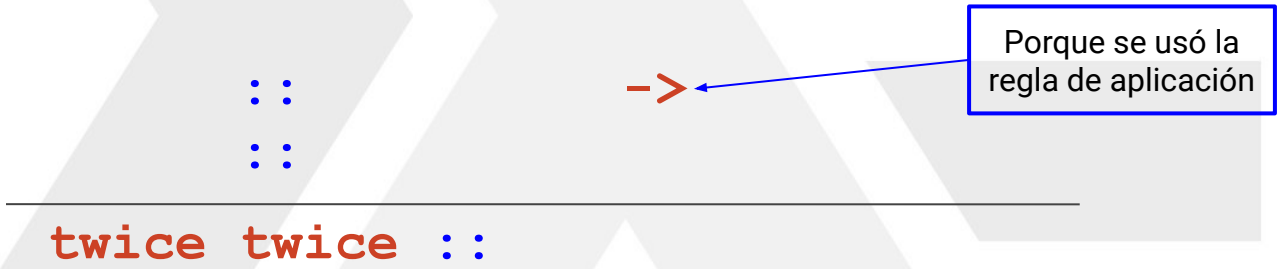
- Ejemplo 5: `twice twice`
  - `twice :: (a -> a) -> (a -> a)`

`twice twice ::`

  - Se plantea la pregunta

Sistema Hindley-Milner polimórfico

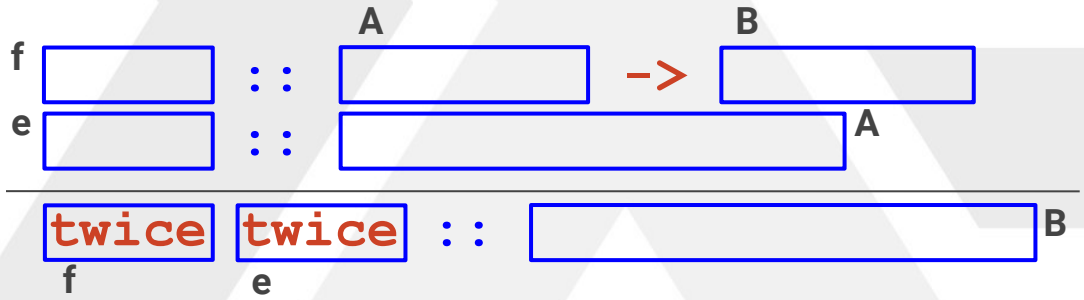
- Ejemplo 5: `twice twice`
  - `twice :: (a -> a) -> (a -> a)`



  - Se determina la regla a aplicar

Sistema Hindley-Milner polimórfico

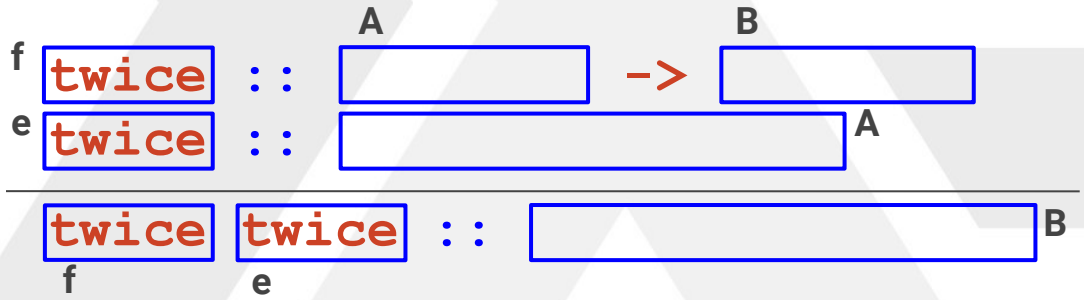
- Ejemplo 5: `twice twice`
  - `twice :: (a -> a) -> (a -> a)`



  - Se analizan cuáles son los componentes

Sistema Hindley-Milner polimórfico

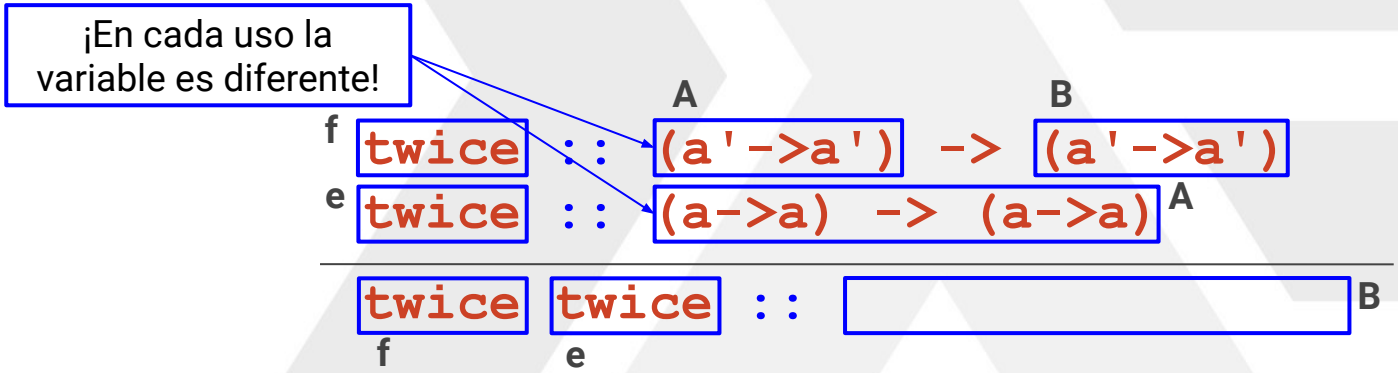
- Ejemplo 5: `twice twice`
  - `twice :: (a -> a) -> (a -> a)`



  - Se comienzan a colocar las piezas en su lugar

Sistema Hindley-Milner polimórfico

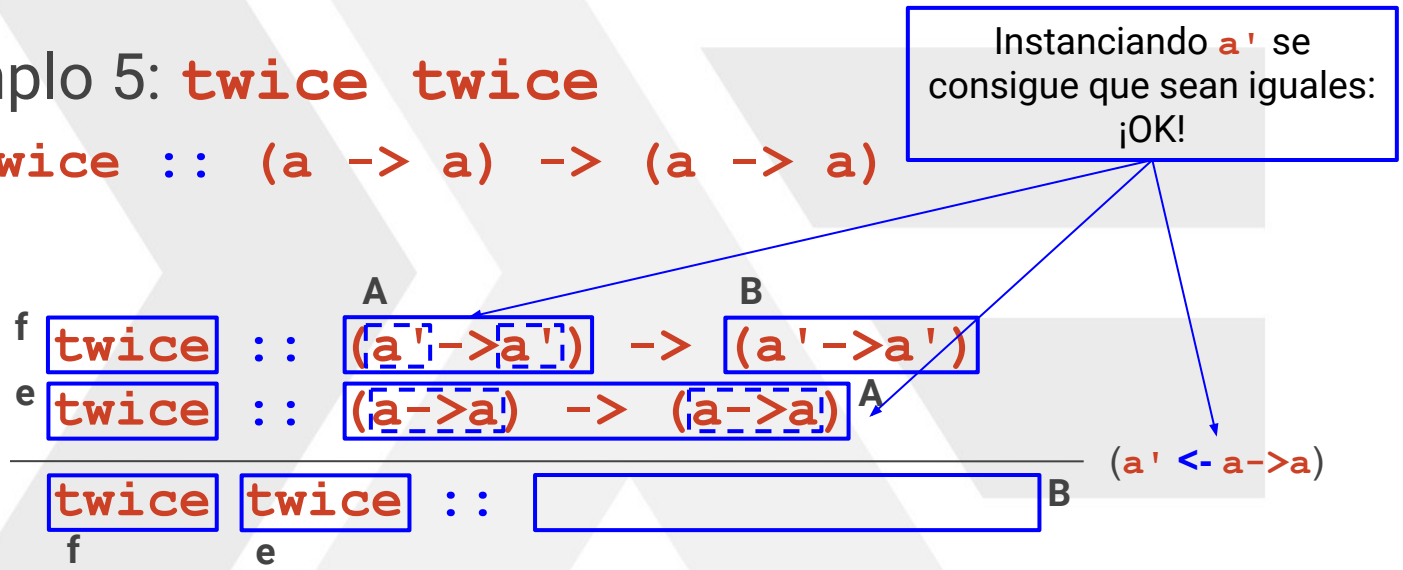
- Ejemplo 5: `twice twice`
  - `twice :: (a -> a) -> (a -> a)`



  - Se resuelven los antecedentes

Sistema Hindley-Milner polimórfico

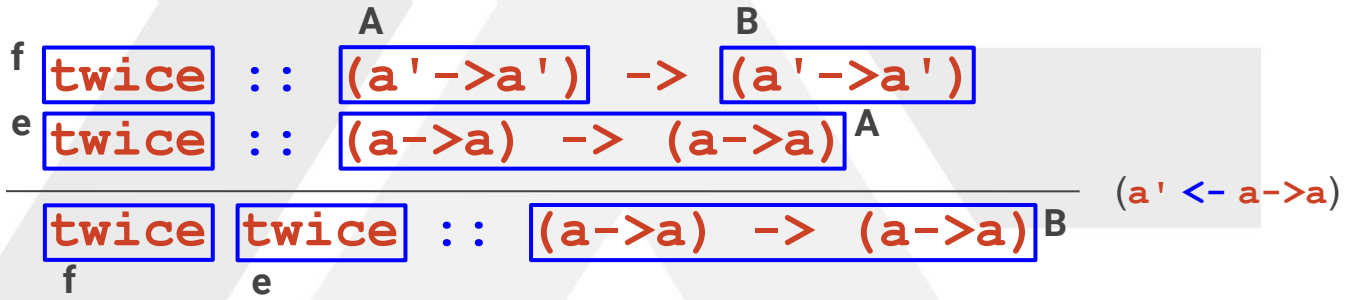
- Ejemplo 5: `twice twice`
  - `twice :: (a -> a) -> (a -> a)`



  - Se busca verificar las condiciones (quizás instanciando variables polimórficas)

Sistema Hindley-Milner polimórfico

- Ejemplo 5: `twice twice`
  - `twice :: (a -> a) -> (a -> a)`



  - Se concluye con el resultado:

```
twice twice :: (a->a) -> (a->a)
```

Sistema Hindley-Milner polimórfico

- En `twice twice` ambos usos tienen distinto tipo
  - La ocurrencia de la derecha

    ```
    twice :: (a -> a) -> (a -> a)
    ```

  - El ocurrencia de la izquierda

    ```
    twice :: ((a->a)->(a->a))
                  -> ((a->a)->(a->a))
    ```

    (`twice :: (a'->a')->(a'->a')` con `a'<-a->a`)

  - Por eso la izquierda puede aplicarse a la derecha (coincide el tipo del parámetro con el del argumento)

Observaciones finales

- Ventajas de un sistema de tipos
  - Detección temprana de errores
  - Documentación rudimentaria
  - Oportunidades de optimización en ejecución
- Existen numerosos sistemas de tipos con diferentes características

Observaciones finales

- Sistema de tipado **fuerte** (*strong typing*)
  - Cuando una expresión tiene tipo,
    NO falla *por problemas de tipos*
  - Ejemplo: Haskell
  - Contrajemplos: C, Java

Resumen

Resumen

- Validez de expresiones
  - Reglas sintácticas
  - Reglas de tipo
- Sistemas de tipos
  - Gran variedad de sistemas y características
    - Sistema H-M (con polimorfismo)
  - Numerosas ventajas
    - P.ej. el tipo ayuda a tener idea del posible comportamiento