Parametric Polymorphism Through Run-time Sealing

or, Theorems for Low, Low Prices!

Amal Ahmed
Lindsey Kuper
Jacob Matthews

Northeastern University
Programming Languages Seminar
February 23, 2011

What is parametricity?

Data abstraction

Data abstraction

Separation of implementation and interface

Data abstraction

Separation of implementation and interface

```
Counter = ∃α. {new : α,
               inc : α → α,
               get : α → Nat}
```

Data abstraction

Separation of implementation and interface

```
Counter = ∃α. {new : α,
               inc : α → α,
               get : α → Nat}
```

```
c1 = {new = 0,
      inc = λx: Nat. x + 1,
      get = λx: Nat. x}

ctr1 = pack Nat, c1 as Counter
```

Data abstraction

Separation of implementation and interface

```
Counter = ∃α. {new : α,
               inc : α → α,
               get : α → Nat}
```

```
c1 = {new = 0,
      inc = λx: Nat. x + 1,
      get = λx: Nat. x}

ctr1 = pack Nat, c1 as Counter
```

```
c2 = {new = 0,
      inc = λx: Int. x - 1,
      get = λx: Int. toNat(0 - x)}

ctr2 = pack Int, c2 as Counter
```

Data abstraction

Separation of implementation and interface

```
Counter = ∃α. {new : α,
               inc : α → α,
               get : α → Nat}
```

```
c1 = {new = 0,
      inc = λx: Nat. x + 1,
      get = λx: Nat. x}

ctr1 = pack Nat, c1 as Counter
```

```
c2 = {new = 0,
      inc = λx: Int. x - 1,
      get = λx: Int. toNat(0 - x)}

ctr2 = pack Int, c2 as Counter
```

indistinguishable

Existential types...

```
c1 = {new = 0,
      inc = λx: Nat. x + 1,
      get = λx: Nat. x}

ctr1 = pack Nat, c1 as Counter
```

```
c2 = {new = 0,
      inc = λx: Int. x - 1,
      get = λx: Int. toNat(0 - x)}

ctr2 = pack Int, c2 as Counter
```

indistinguishable

Existential types...

```
c1 = {new = 0,
      inc = λx: Nat. x + 1,
      get = λx: Nat. x}

ctr1 = pack Nat, c1 as Counter
```

```
c2 = {new = 0,
      inc = λx: Int. x - 1,
      get = λx: Int. toNat(0 - x)}

ctr2 = pack Int, c2 as Counter
```

indistinguishable

- If two expressions have the same existential type, no program context can distinguish them.

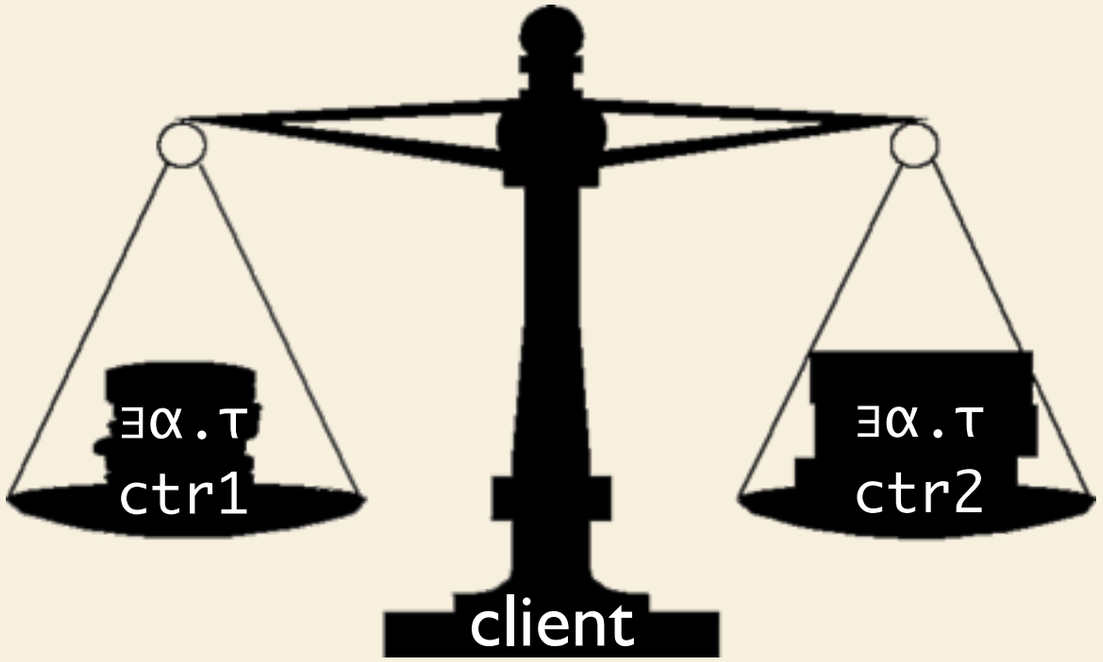
Existential types...

```
c1 = {new = 0,
      inc = λx: Nat. x + 1,
      get = λx: Nat. x}

ctr1 = pack Nat, c1 as Counter
```

```
c2 = {new = 0,
      inc = λx: Int. x - 1,
      get = λx: Int. toNat(0 - x)}

ctr2 = pack Int, c2 as Counter
```

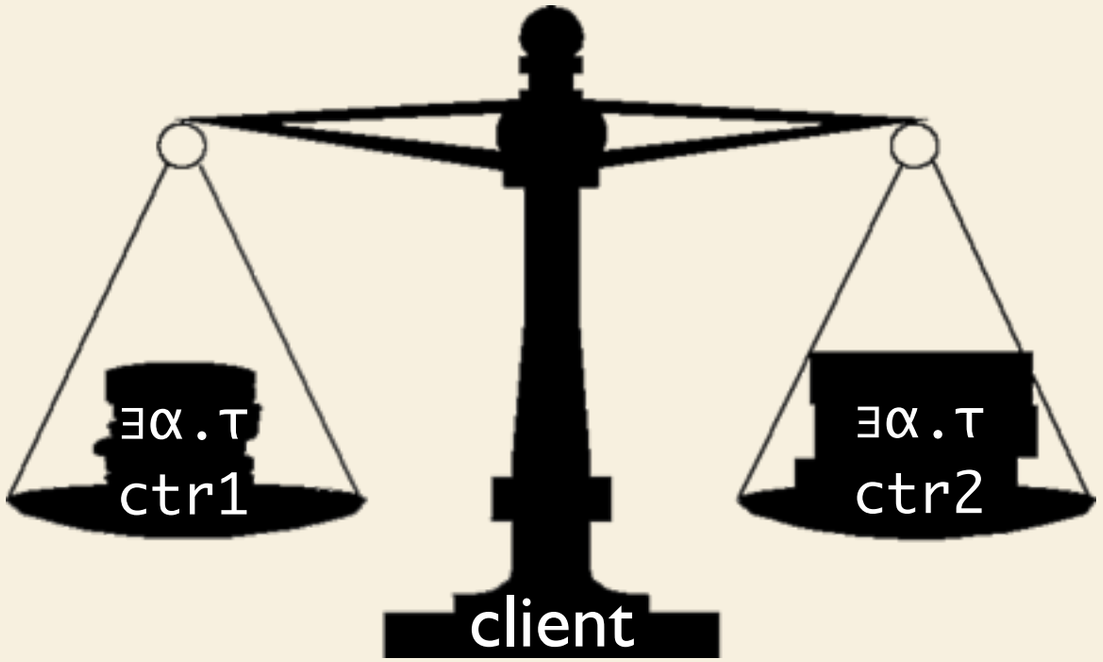
indistinguishable

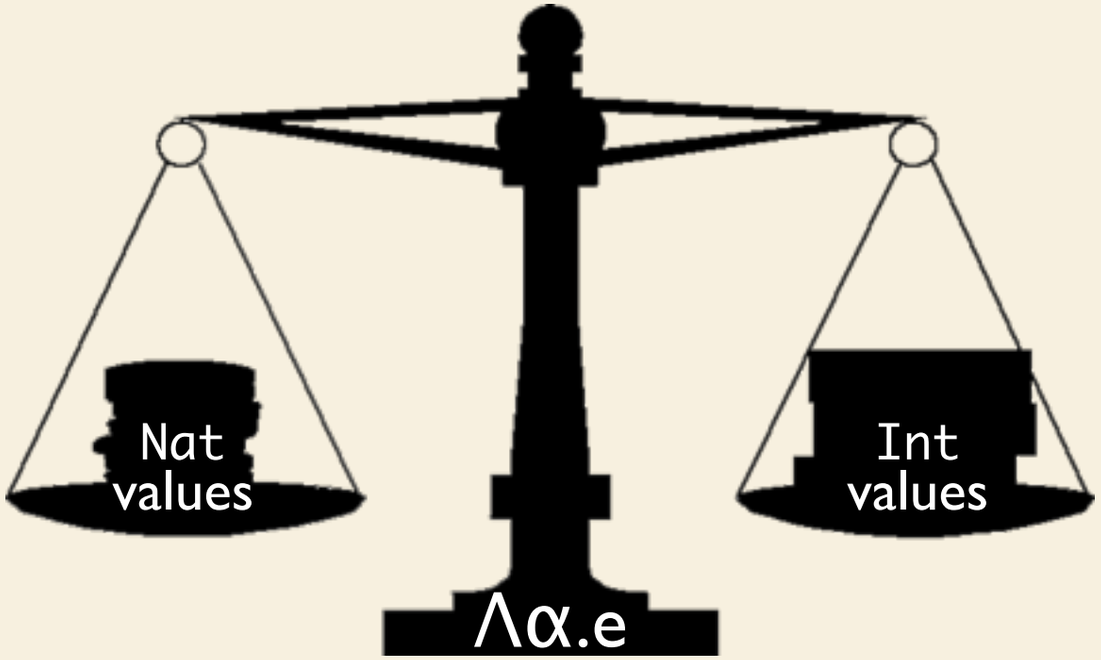
- If two expressions have the same existential type, no program context can distinguish them.



Existential types...and their dual, universal types

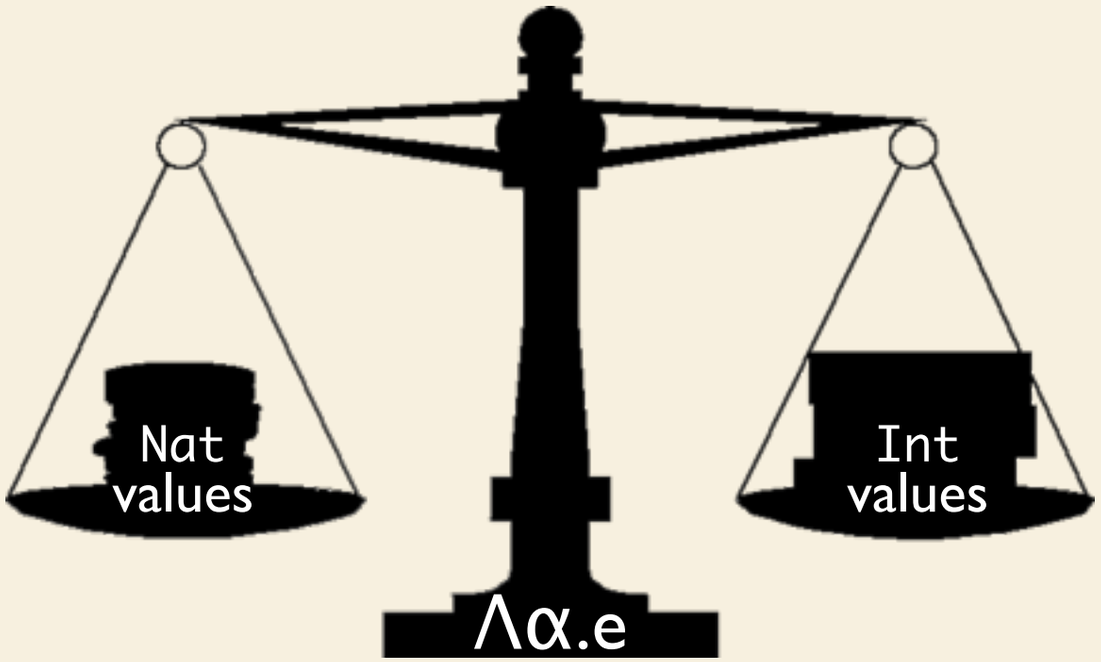
- If two expressions have the same existential type, no program context can distinguish them.





- No two program contexts (*instantiations*) can cause an expression of type ∀α.τ to behave differently.

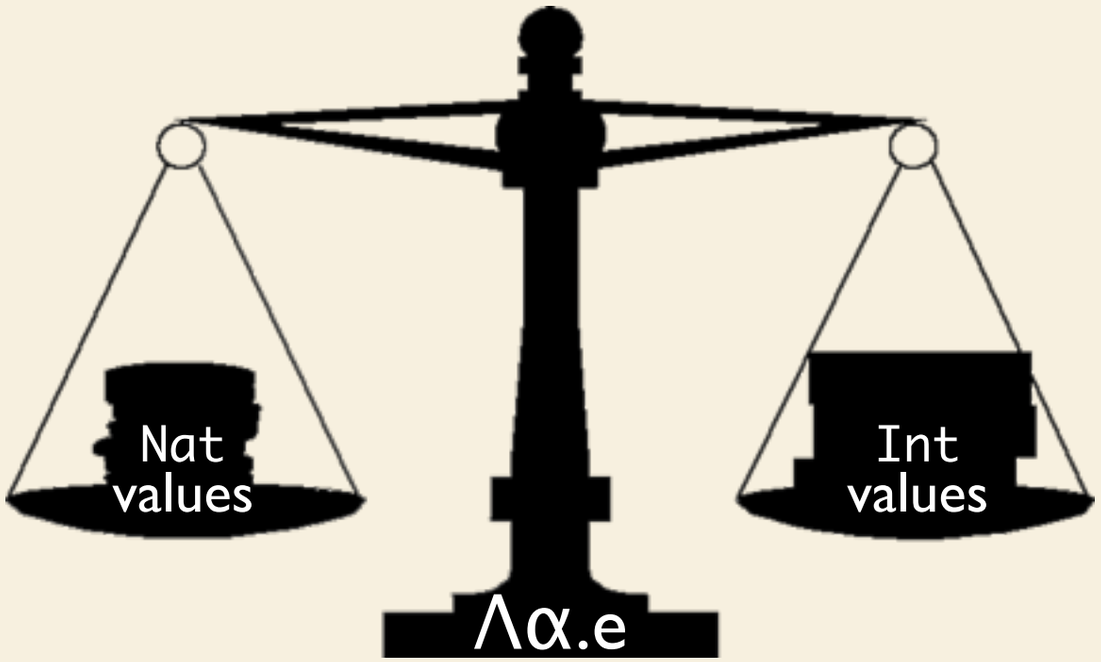
...and their dual, universal types



- No two program contexts (*instantiations*) can cause an expression of type ∀α.τ to behave differently.

...and their dual, universal types

```
f : ∀α. α → α
```

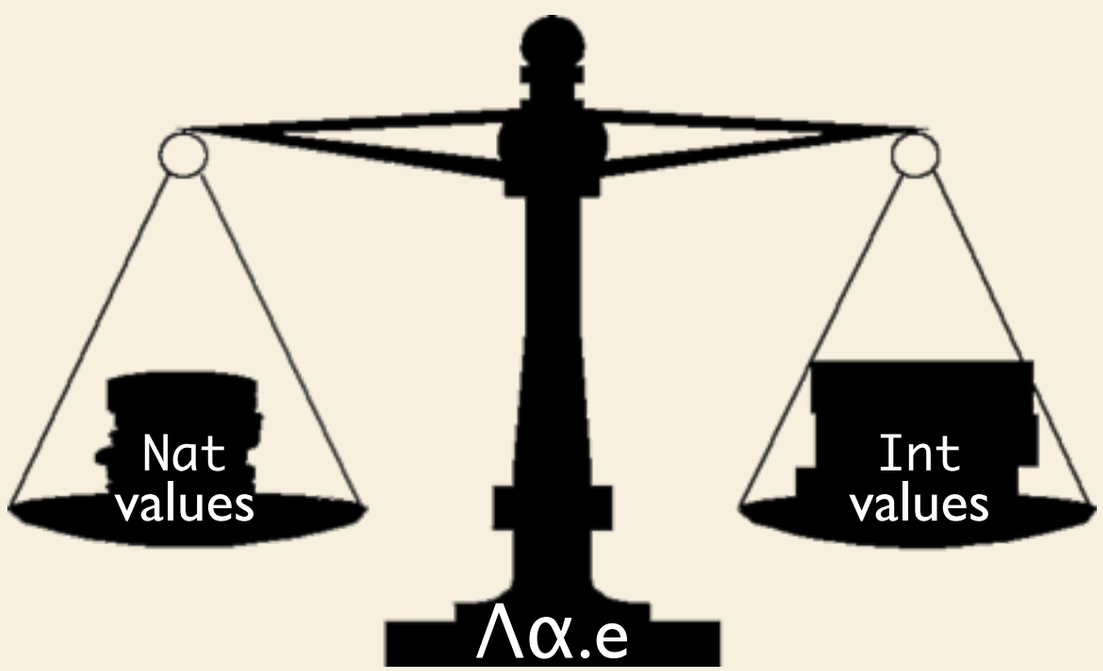


- No two program contexts (*instantiations*) can cause an expression of type ∀α.τ to behave differently.

...and their dual, universal types

```
f : ∀α. α → α
```

```
Nat
```

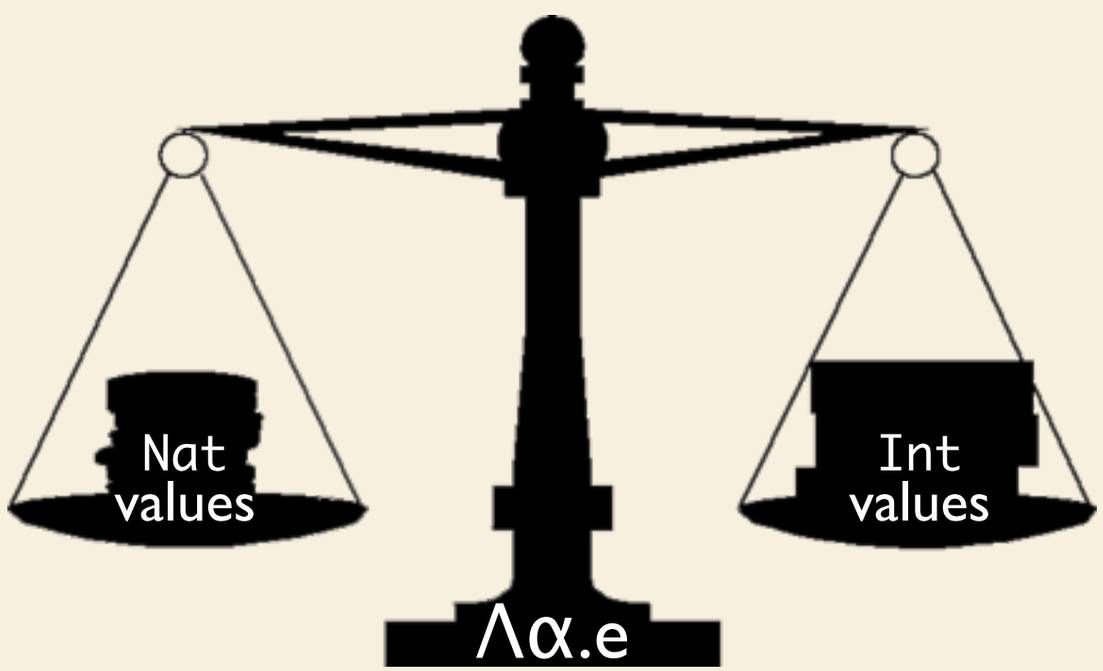


- No two program contexts (*instantiations*) can cause an expression of type ∀α.τ to behave differently.

...and their dual, universal types

```
f : ∀α. α → α
```

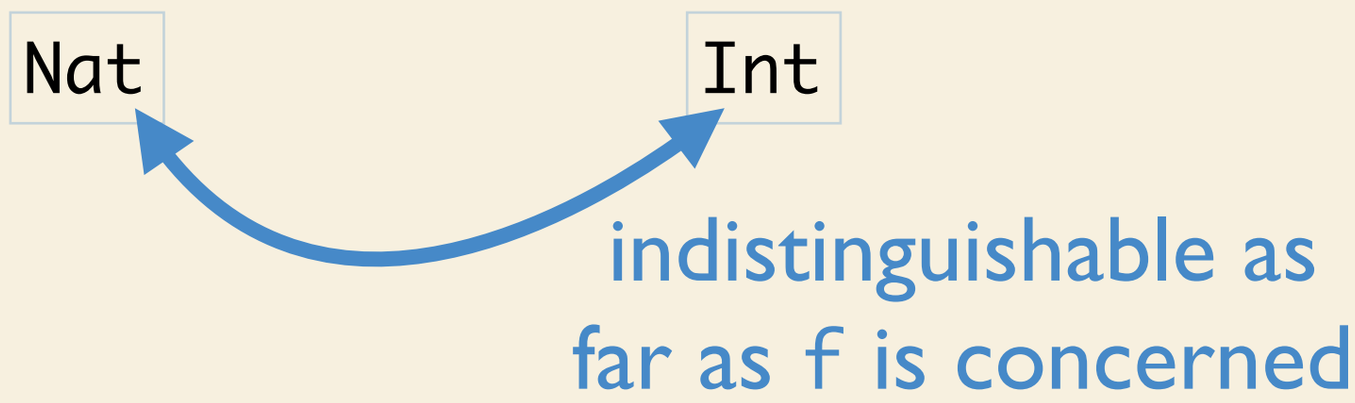
`Nat` `Int`

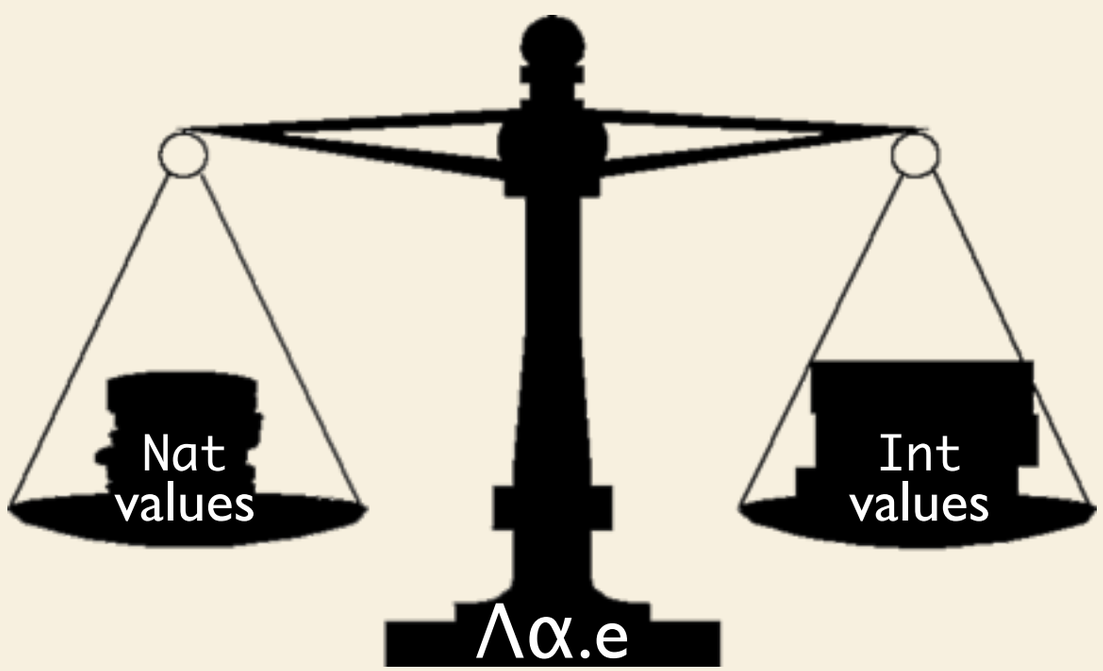


- No two program contexts (*instantiations*) can cause an expression of type ∀α.τ to behave differently.

...and their dual, universal types

```
f : ∀α. α → α
```

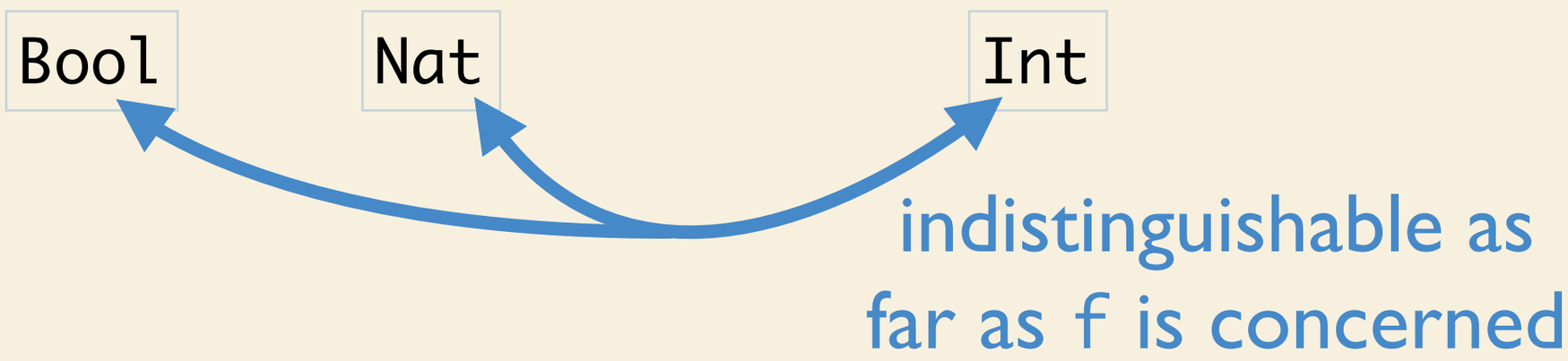


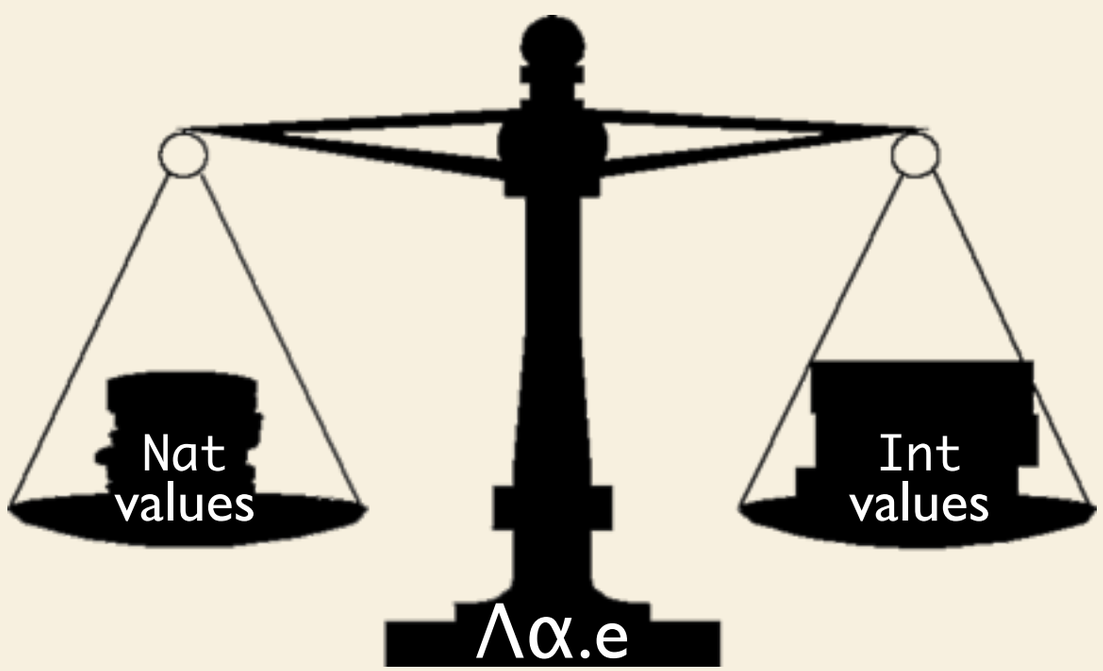


- No two program contexts (*instantiations*) can cause an expression of type ∀α.τ to behave differently.

...and their dual, universal types

```
f : ∀α. α → α
```

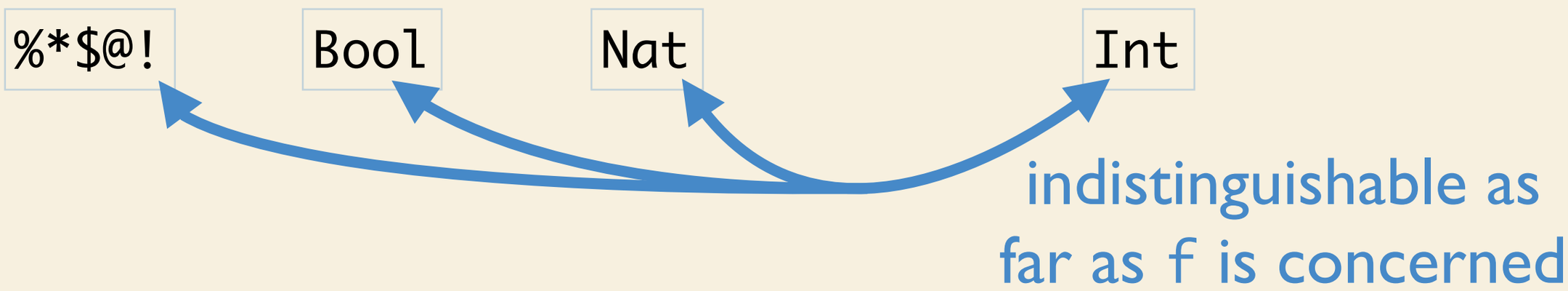


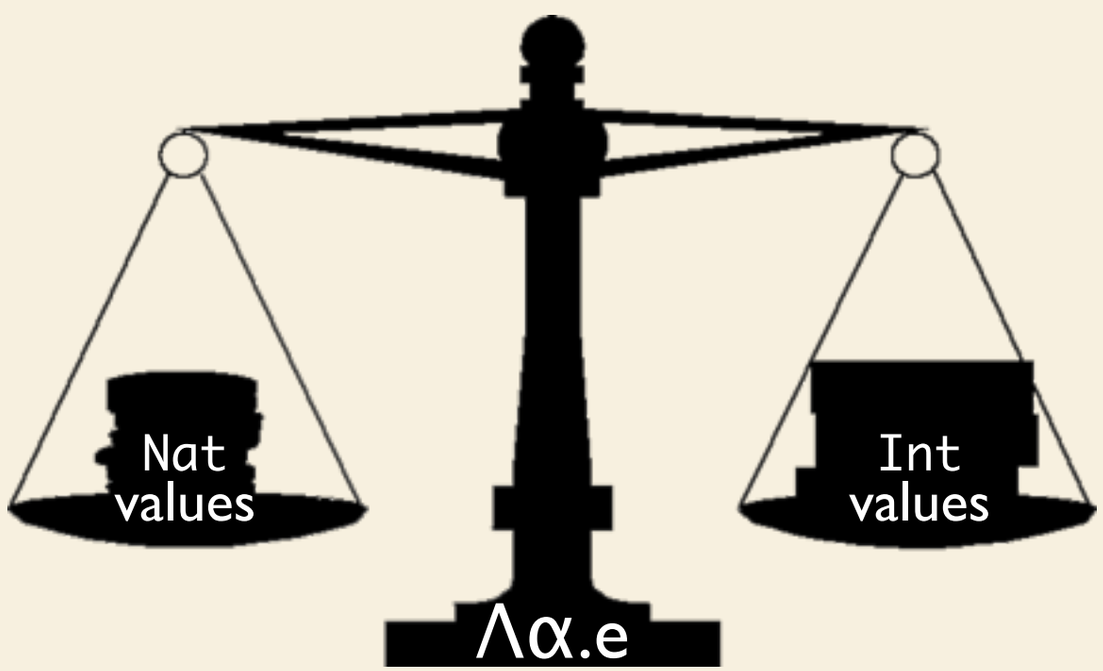


- No two program contexts (*instantiations*) can cause an expression of type ∀α.τ to behave differently.

...and their dual, universal types

```
f : ∀α. α → α
```

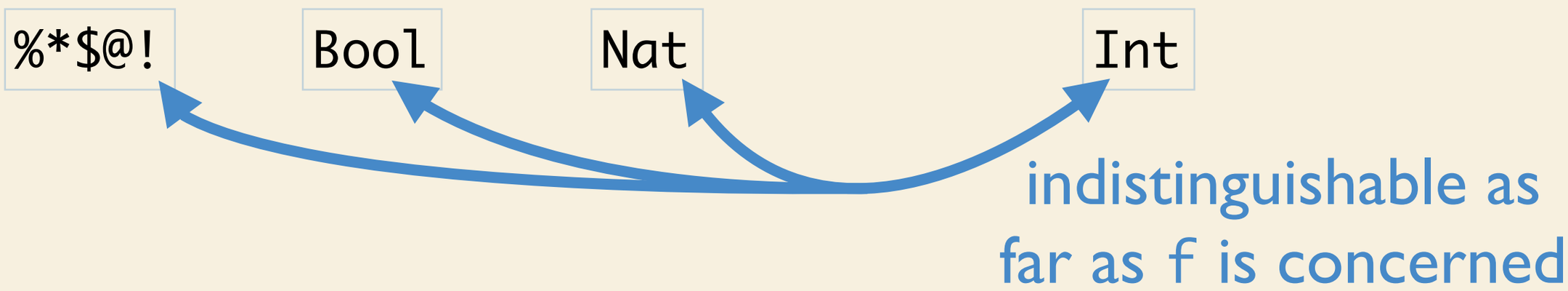


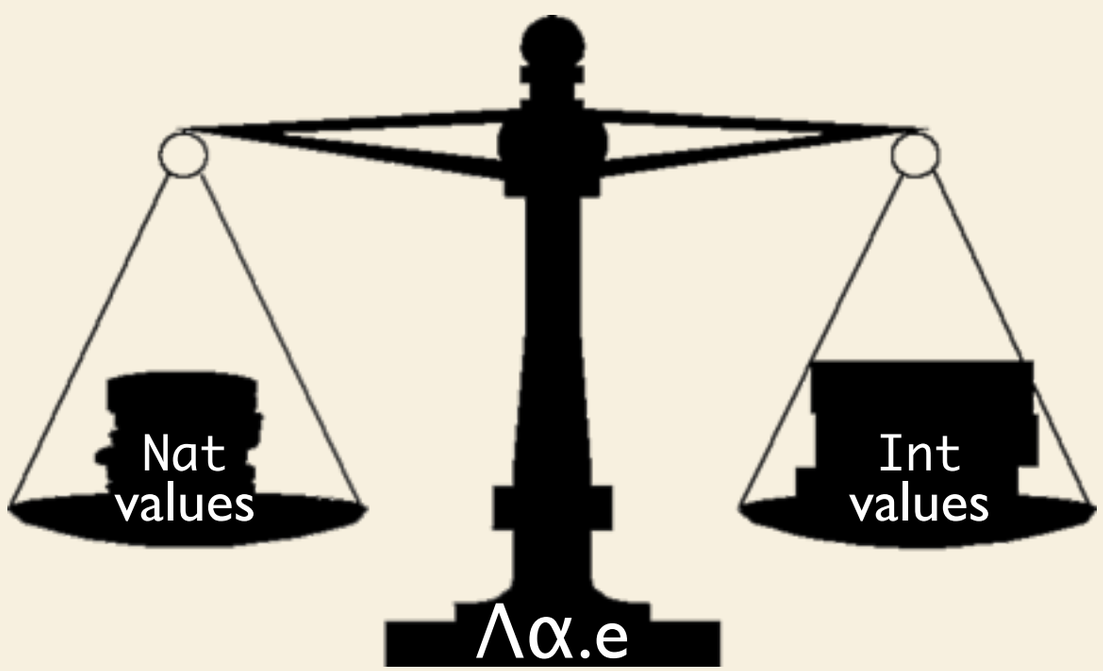


- No two program contexts (*instantiations*) can cause an expression of type ∀α.τ to behave differently.

...and their dual, universal types

```
f : ∀α. α → α
f = Λα. λx: α. x
```

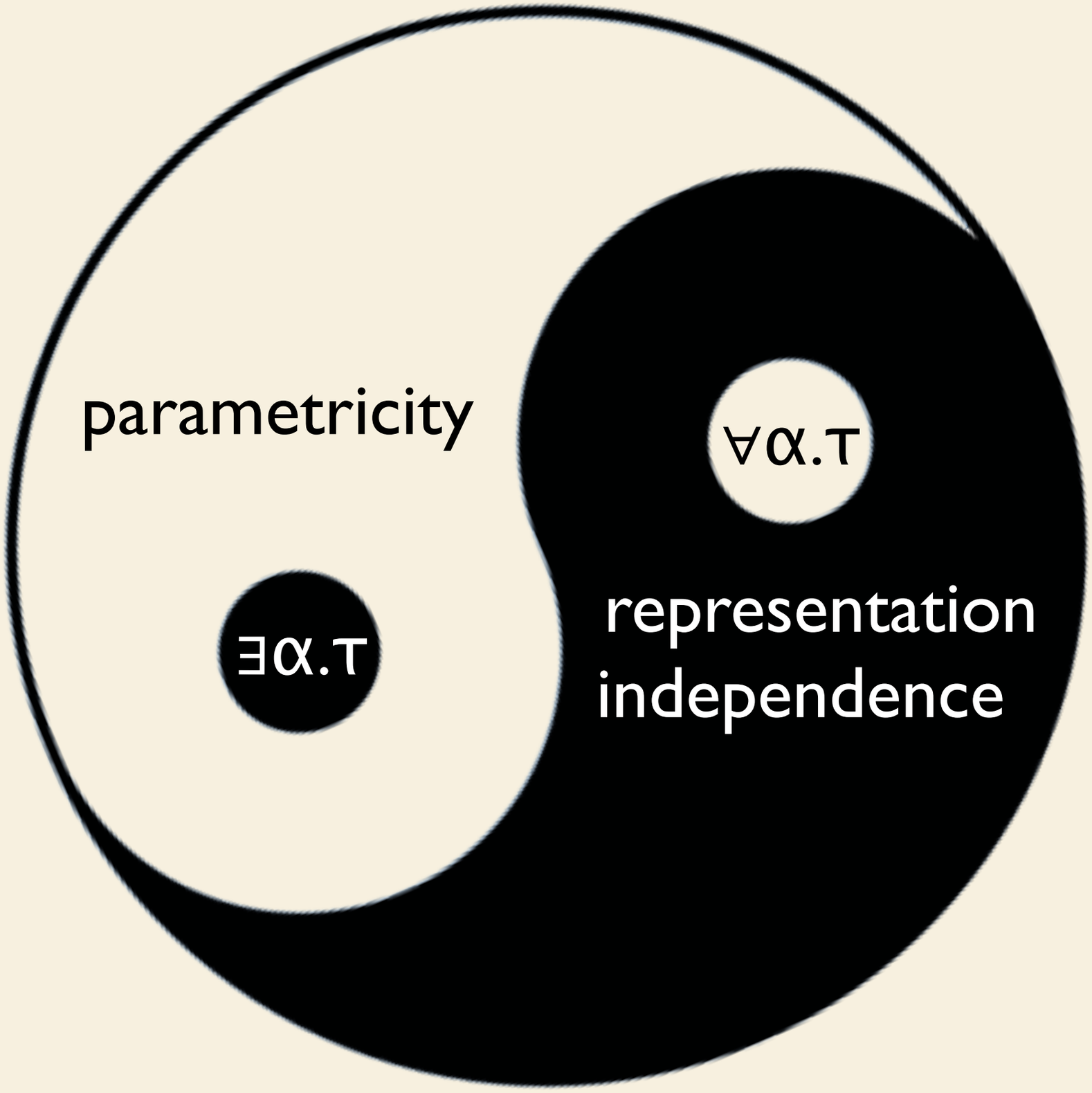




- No two program contexts (*instantiations*) can cause an expression of type ∀α.τ to behave differently.

Existential types...and their dual, universal types

Existential types...and their dual, universal types



Breaking parametricity

How to break parametricity in one easy step

```
Λα. λx: α. (if (nat? x)
               (+ x 1)
               x)
```

How to break parametricity in one easy step

```
Λα. λx: α. (if (nat? x)
                (+ x 1)
                x)
```

behaves differently at run-time depending on how α is instantiated

How to break parametricity in one easy step

```
Λα. λx: α. (if (nat? x)
                (+ x 1)
                x)
```

behaves differently at run-time depending on how α is instantiated

Putting dynamically typed code in an otherwise statically typed program provides a way to

"smuggle values past the type system"

(Abadi *et al.*, 1989)

A two-language system

A two-language system

- How can we assign a type to a program that's written in two languages?

A two-language system

- How can we assign a type to a program that's written in two languages?
- We'll combine a minimal "Scheme" and a minimal "ML" in a multi-language embedding (Matthews & Findler, 2007):

A two-language system

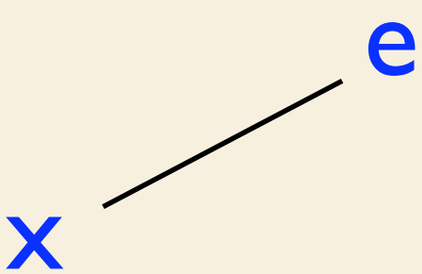
- How can we assign a type to a program that's written in two languages?
- We'll combine a minimal "Scheme" and a minimal "ML" in a multi-language embedding (Matthews & Findler, 2007):

e

e

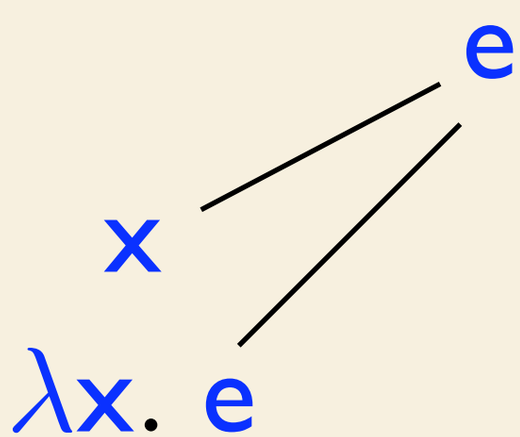
A two-language system

- How can we assign a type to a program that's written in two languages?
- We'll combine a minimal "Scheme" and a minimal "ML" in a multi-language embedding (Matthews & Findler, 2007):



A two-language system

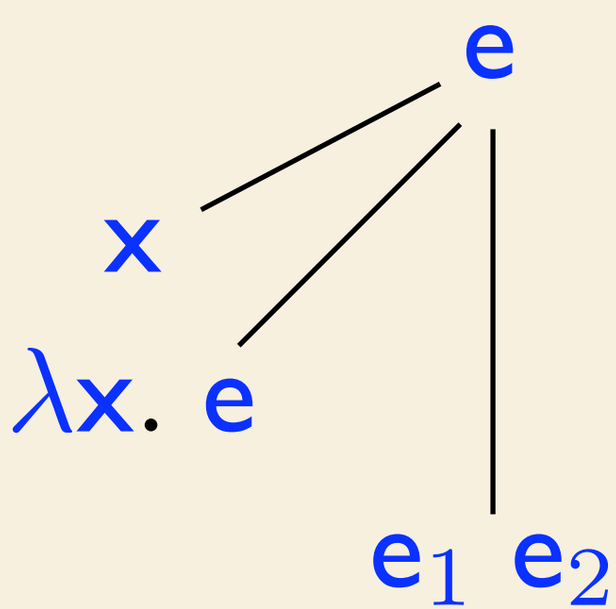
- How can we assign a type to a program that's written in two languages?
- We'll combine a minimal "Scheme" and a minimal "ML" in a multi-language embedding (Matthews & Findler, 2007):



e

A two-language system

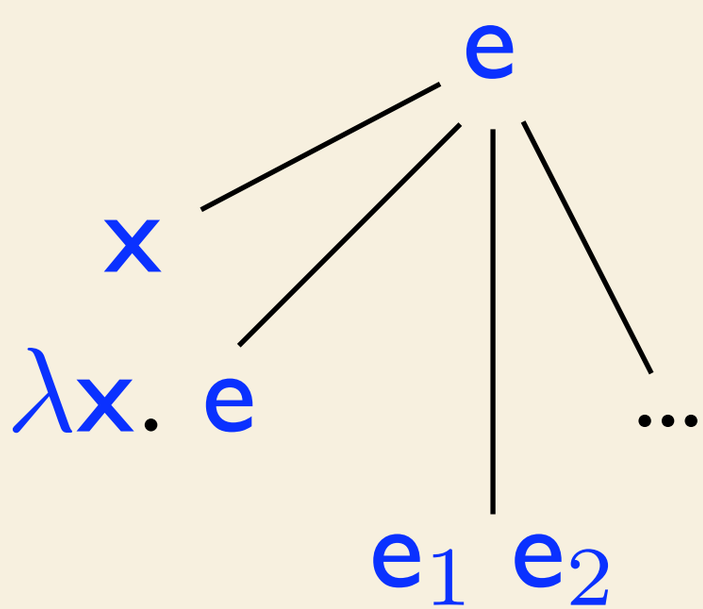
- How can we assign a type to a program that's written in two languages?
- We'll combine a minimal "Scheme" and a minimal "ML" in a multi-language embedding (Matthews & Findler, 2007):



e

A two-language system

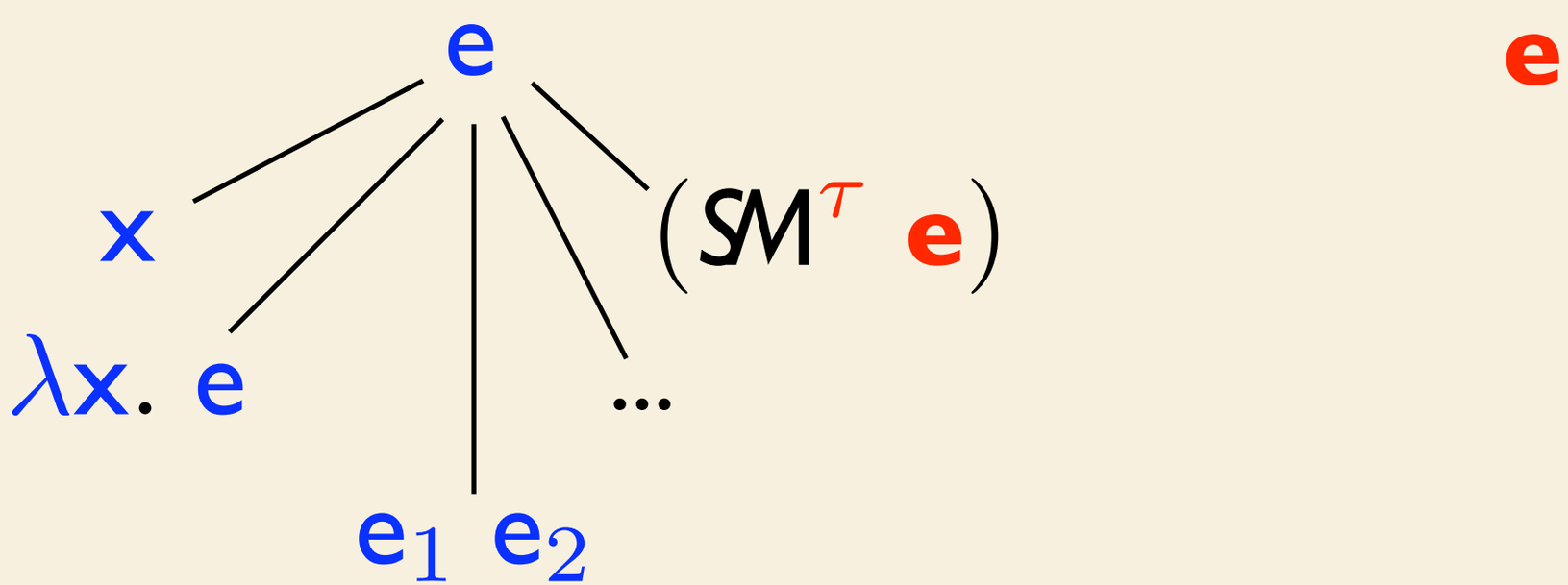
- How can we assign a type to a program that's written in two languages?
- We'll combine a minimal "Scheme" and a minimal "ML" in a multi-language embedding (Matthews & Findler, 2007):



e

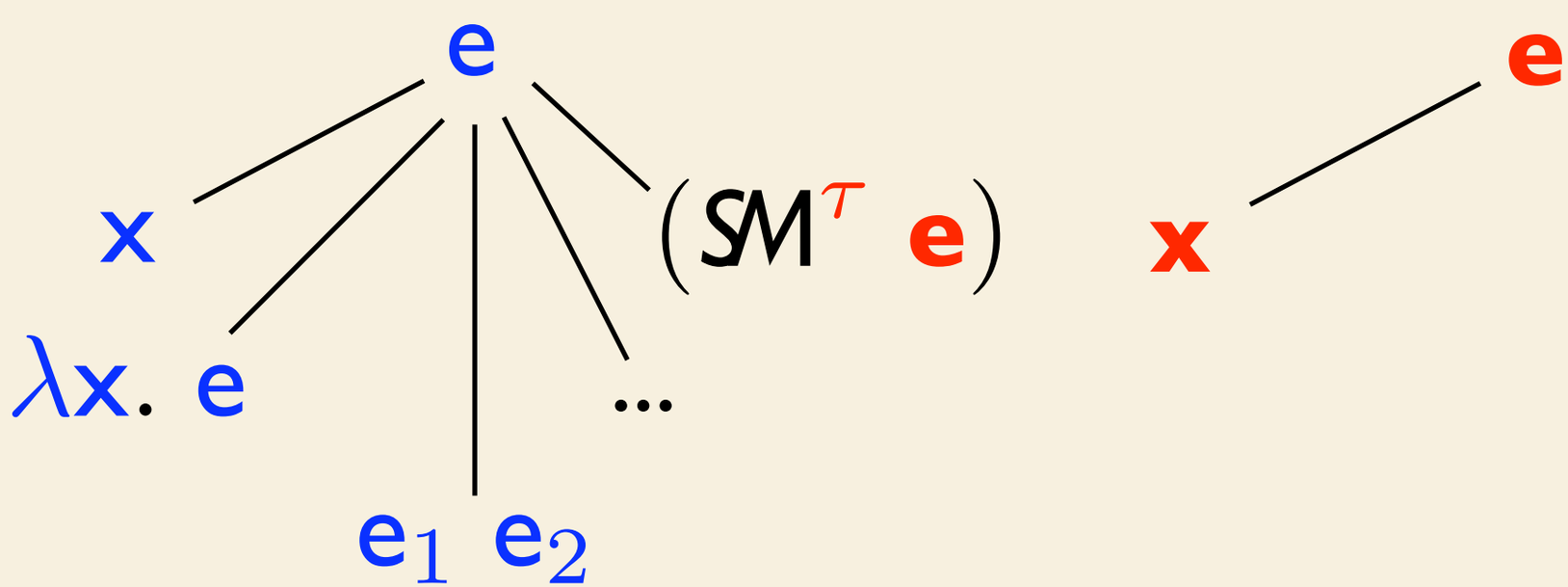
A two-language system

- How can we assign a type to a program that's written in two languages?
- We'll combine a minimal "Scheme" and a minimal "ML" in a multi-language embedding (Matthews & Findler, 2007):



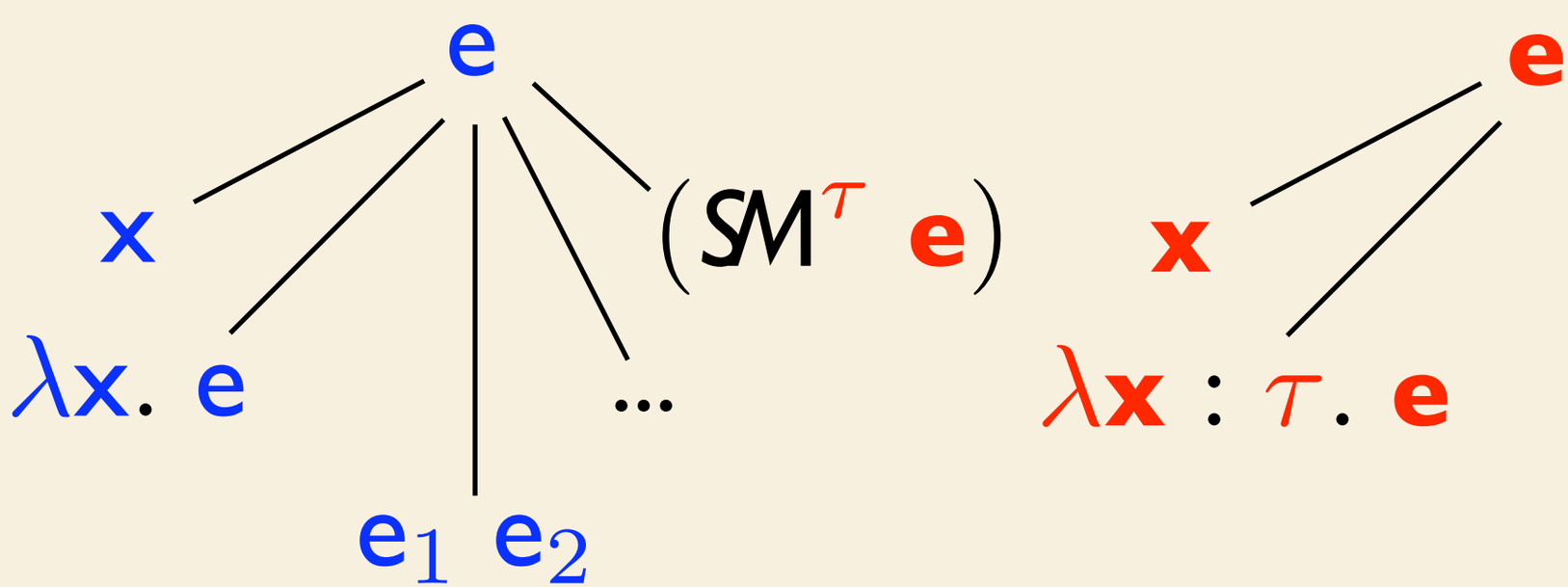
A two-language system

- How can we assign a type to a program that's written in two languages?
- We'll combine a minimal "Scheme" and a minimal "ML" in a multi-language embedding (Matthews & Findler, 2007):



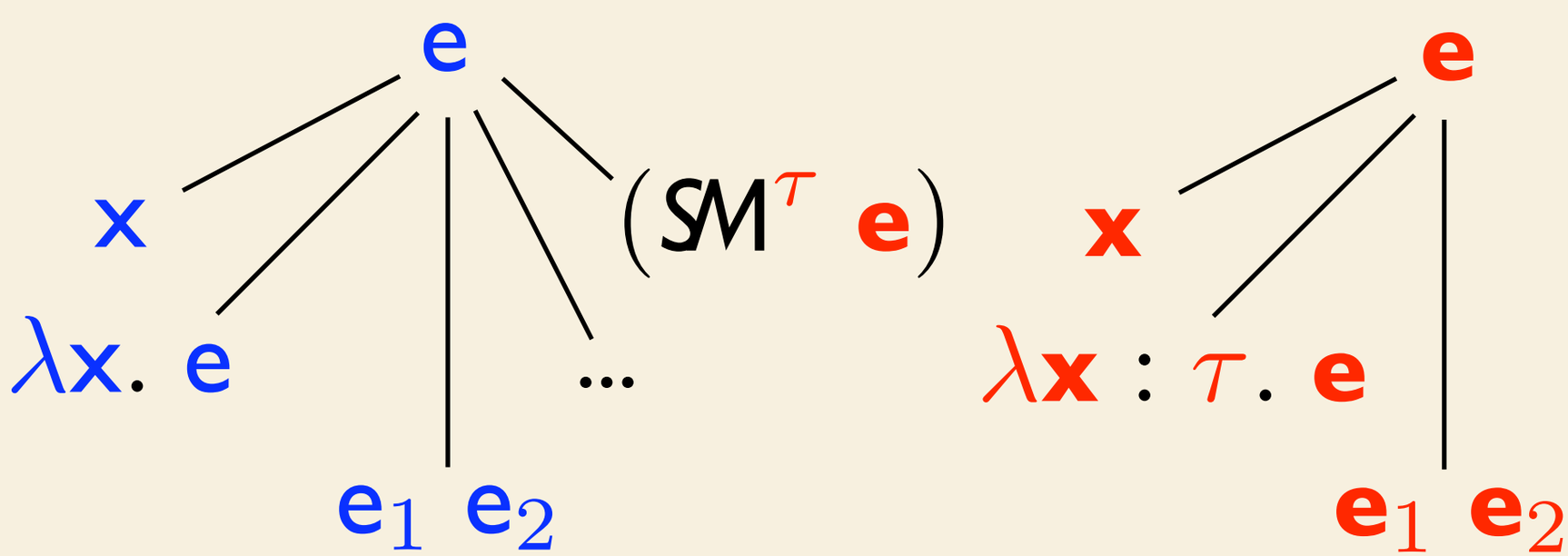
A two-language system

- How can we assign a type to a program that's written in two languages?
- We'll combine a minimal "Scheme" and a minimal "ML" in a multi-language embedding (Matthews & Findler, 2007):



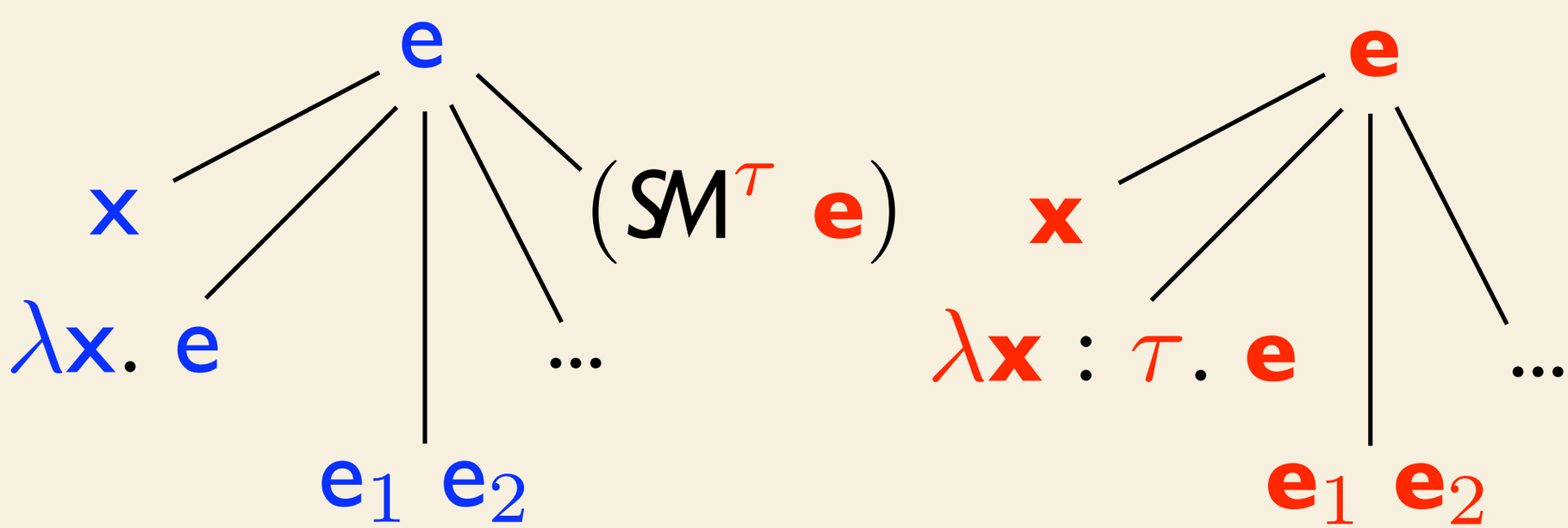
A two-language system

- How can we assign a type to a program that's written in two languages?
- We'll combine a minimal "Scheme" and a minimal "ML" in a multi-language embedding (Matthews & Findler, 2007):



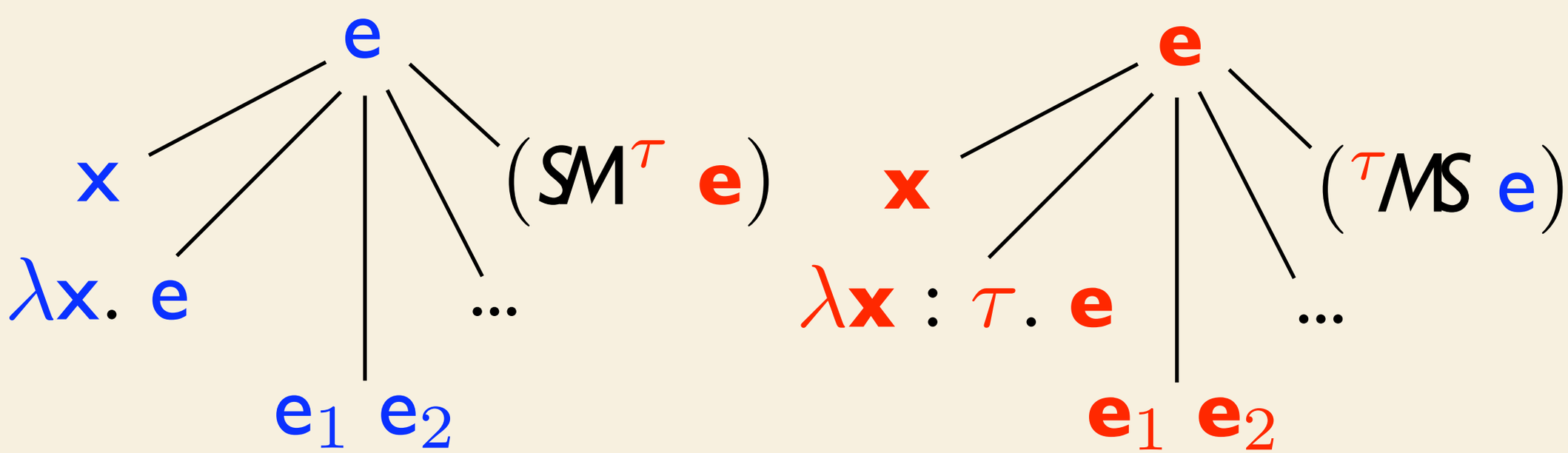
A two-language system

- How can we assign a type to a program that's written in two languages?
- We'll combine a minimal "Scheme" and a minimal "ML" in a multi-language embedding (Matthews & Findler, 2007):



A two-language system

- How can we assign a type to a program that's written in two languages?
- We'll combine a minimal "Scheme" and a minimal "ML" in a multi-language embedding (Matthews & Findler, 2007):



Using a Scheme procedure in ML

$$({}^{\tau_1 \to \tau_2}\mathit{MS}\ (\lambda \mathsf{x}.\ \mathsf{e}))$$

Using a Scheme procedure in ML

have to choose some type at which to embed the procedure

$$({}^{\tau_1 \to \tau_2}\mathit{MS}\ (\lambda \mathsf{x}.\ \mathsf{e}))$$

Using a Scheme procedure in ML

have to choose some type at
which to embed the procedure

$$({}^{\tau_1 \to \tau_2}\mathit{MS}\ (\lambda \mathbf{x}.\ \mathsf{e})) \longmapsto (\lambda \mathbf{x} : \tau_1 . \qquad)$$

Using a Scheme procedure in ML

have to choose some type at
which to embed the procedure

$$({}^{\tau_1 \to \tau_2}\mathit{MS}\ (\lambda \mathbf{x}.\ \mathsf{e})) \longmapsto (\lambda \mathbf{x} : \tau_1.\ ({}^{\tau_2}\mathit{MS}\ (\lambda \mathbf{x}.\ \mathsf{e})\ (\mathit{SM}^{\tau_1}\ \mathbf{x})))$$

Using a Scheme procedure in ML

have to choose some type at
which to embed the procedure

$$({}^{\tau_1 \to \tau_2}\mathit{MS}\ (\lambda \mathbf{x}.\ \mathsf{e})) \longmapsto (\lambda \mathbf{x} : \tau_1.\ ({}^{\tau_2}\mathit{MS}\ (\lambda \mathbf{x}.\ \mathsf{e})\ (\mathit{SM}^{\tau_1}\ \mathbf{x})))$$

direction of conversion reverses
for arguments

A first attempt at polymorphism

$$({}^{\forall\alpha.\ \tau}\mathit{MS}\ (\lambda \mathsf{x}.\ \mathsf{e}))$$

A first attempt at polymorphism

embedding a Scheme procedure in ML at a universal type

$$({}^{\forall\alpha.\ \tau}\mathit{MS}\ (\lambda \mathsf{x}.\ \mathsf{e}))$$

A first attempt at polymorphism

embedding a Scheme procedure in
ML at a universal type

$$({}^{\forall\alpha.\ \tau}\mathit{MS}\ (\lambda \mathsf{x}.\ \mathsf{e})) \quad \longmapsto \quad (\Lambda\alpha. \qquad\qquad\qquad\qquad)$$

A first attempt at polymorphism

embedding a Scheme procedure in
ML at a universal type

$$({}^{\forall\alpha.\ \tau}\mathit{MS}\ (\lambda \mathsf{x}.\ \mathsf{e})) \quad \longmapsto \quad (\Lambda\alpha.\ ({}^{\tau}\mathit{MS}\ (\lambda \mathsf{x}.\ \mathsf{e})))$$

A first attempt at polymorphism

embedding a Scheme procedure in
ML at a universal type

$$({}^{\forall\alpha.\ \tau}\mathit{MS}\ (\lambda \mathsf{x}.\ \mathsf{e})) \quad \longmapsto \quad (\Lambda\alpha.\ ({}^{\tau}\mathit{MS}\ (\lambda \mathsf{x}.\ \mathsf{e})))$$

evaluation stops here, and continues
when we apply to a concrete type:

$$(\Lambda\alpha.\ \mathbf{e})\ \mathbf{Nat} \longmapsto \mathbf{e}[\alpha := \mathbf{Nat}]$$

A first attempt at polymorphism: example

$$({}^{\forall\alpha.\ \alpha\to\alpha}\mathit{MS}\ (\lambda \mathbf{x}.\ \mathbf{x}))\ \mathbf{Nat}\ \overline{3}$$

A first attempt at polymorphism: example

$$(^{\forall\alpha.\ \alpha\to\alpha}\mathit{MS}\ (\lambda \mathbf{x}.\ \mathbf{x}))\ \mathbf{Nat}\ \overline{3}$$

$$\longrightarrow \quad (\Lambda\alpha.\ (^{\alpha\to\alpha}\mathit{MS}\ (\lambda \mathbf{x}.\ \mathbf{x}))\ \mathbf{Nat}\ \overline{3}$$

A first attempt at polymorphism: example

$$
\begin{array}{ll}
 & ({}^{\forall\alpha.\ \alpha\to\alpha}\mathit{MS}\ (\lambda \mathbf{x}.\ \mathbf{x}))\ \mathbf{Nat}\ \overline{3} \\
\longrightarrow & (\Lambda\alpha.\ ({}^{\alpha\to\alpha}\mathit{MS}\ (\lambda \mathbf{x}.\ \mathbf{x}))\ \mathbf{Nat}\ \overline{3} \\
\longrightarrow & ({}^{\mathbf{Nat}\to\mathbf{Nat}}\mathit{MS}\ (\lambda \mathbf{x}.\ \mathbf{x}))\ \overline{3}
\end{array}
$$

A first attempt at polymorphism: example

$$
\begin{aligned}
& ({}^{\forall\alpha.\ \alpha\to\alpha}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x}))\ \mathbf{Nat}\ \overline{3} \\
\longrightarrow\quad & (\Lambda\alpha.\ ({}^{\alpha\to\alpha}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x}))\ \mathbf{Nat}\ \overline{3} \\
\longrightarrow\quad & ({}^{\mathbf{Nat}\to\mathbf{Nat}}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x}))\ \overline{3} \\
\longrightarrow\quad & (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\mathbf{Nat}}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x})\ (\mathsf{SM}^{\mathbf{Nat}}\ \mathbf{y})))\ \overline{3}
\end{aligned}
$$

A first attempt at polymorphism: example

$$
\begin{aligned}
& ({}^{\forall\alpha.\ \alpha\to\alpha}\mathit{MS}\ (\lambda \mathbf{x}.\ \mathbf{x}))\ \mathbf{Nat}\ \overline{3} \\
\longrightarrow\quad & (\Lambda\alpha.\ ({}^{\alpha\to\alpha}\mathit{MS}\ (\lambda \mathbf{x}.\ \mathbf{x}))\ \mathbf{Nat}\ \overline{3} \\
\longrightarrow\quad & ({}^{\mathbf{Nat}\to\mathbf{Nat}}\mathit{MS}\ (\lambda \mathbf{x}.\ \mathbf{x}))\ \overline{3} \\
\longrightarrow\quad & (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\mathbf{Nat}}\mathit{MS}\ (\lambda \mathbf{x}.\ \mathbf{x})\ (\mathit{SM}^{\mathbf{Nat}}\ \mathbf{y})))\ \overline{3} \\
\longrightarrow\quad & ({}^{\mathbf{Nat}}\mathit{MS}\ (\lambda \mathbf{x}.\ \mathbf{x})\ (\mathit{SM}^{\mathbf{Nat}}\ \overline{3}))
\end{aligned}
$$

A first attempt at polymorphism: example

$$
\begin{array}{ll}
 & ({}^{\forall\alpha.\ \alpha\to\alpha}\mathit{MS}\ (\lambda \mathbf{x}.\ \mathbf{x}))\ \mathbf{Nat}\ \overline{3} \\
\longrightarrow & (\Lambda\alpha.\ ({}^{\alpha\to\alpha}\mathit{MS}\ (\lambda \mathbf{x}.\ \mathbf{x}))\ \mathbf{Nat}\ \overline{3} \\
\longrightarrow & ({}^{\mathbf{Nat}\to\mathbf{Nat}}\mathit{MS}\ (\lambda \mathbf{x}.\ \mathbf{x}))\ \overline{3} \\
\longrightarrow & (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\mathbf{Nat}}\mathit{MS}\ (\lambda \mathbf{x}.\ \mathbf{x})\ (\mathit{SM}^{\mathbf{Nat}}\ \mathbf{y})))\ \overline{3} \\
\longrightarrow & ({}^{\mathbf{Nat}}\mathit{MS}\ (\lambda \mathbf{x}.\ \mathbf{x})\ (\mathit{SM}^{\mathbf{Nat}}\ \overline{3}))
\end{array}
$$

first-order values are assumed to be convertible

A first attempt at polymorphism: example

$$
\begin{aligned}
& ({}^{\forall\alpha.\ \alpha\to\alpha}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x}))\ \mathbf{Nat}\ \overline{3} \\
\longrightarrow\quad & (\Lambda\alpha.\ ({}^{\alpha\to\alpha}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x}))\ \mathbf{Nat}\ \overline{3} \\
\longrightarrow\quad & ({}^{\mathbf{Nat}\to\mathbf{Nat}}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x}))\ \overline{3} \\
\longrightarrow\quad & (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\mathbf{Nat}}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x})\ (\mathsf{SM}^{\mathbf{Nat}}\ \mathbf{y})))\ \overline{3} \\
\longrightarrow\quad & ({}^{\mathbf{Nat}}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x})\ (\mathsf{SM}^{\mathbf{Nat}}\ \overline{3})) \\
\longrightarrow\quad & ({}^{\mathbf{Nat}}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x})\ \overline{3})
\end{aligned}
$$

first-order values are assumed to be convertible

A first attempt at polymorphism: example

$$
\begin{array}{ll}
 & ({}^{\forall\alpha.\ \alpha\to\alpha}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x}))\ \mathbf{Nat}\ \overline{3} \\
\longrightarrow & (\Lambda\alpha.\ ({}^{\alpha\to\alpha}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x}))\ \mathbf{Nat}\ \overline{3} \\
\longrightarrow & ({}^{\mathbf{Nat}\to\mathbf{Nat}}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x}))\ \overline{3} \\
\longrightarrow & (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\mathbf{Nat}}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x})\ (\mathsf{SM}^{\mathbf{Nat}}\ \mathbf{y})))\ \overline{3} \\
\longrightarrow & ({}^{\mathbf{Nat}}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x})\ (\mathsf{SM}^{\mathbf{Nat}}\ \overline{3})) \\
\longrightarrow & ({}^{\mathbf{Nat}}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x})\ \overline{3}) \\
\longrightarrow & ({}^{\mathbf{Nat}}\mathsf{MS}\ \overline{3})
\end{array}
$$

first-order values are assumed to be convertible

A first attempt at polymorphism: example

$$
\begin{array}{ll}
 & ({}^{\forall\alpha.\ \alpha\to\alpha}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x}))\ \mathbf{Nat}\ \overline{3} \\
\longrightarrow & (\Lambda\alpha.\ ({}^{\alpha\to\alpha}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x}))\ \mathbf{Nat}\ \overline{3} \\
\longrightarrow & ({}^{\mathbf{Nat}\to\mathbf{Nat}}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x}))\ \overline{3} \\
\longrightarrow & (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\mathbf{Nat}}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x})\ (\mathsf{SM}^{\mathbf{Nat}}\ \mathbf{y})))\ \overline{3} \\
\longrightarrow & ({}^{\mathbf{Nat}}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x})\ (\mathsf{SM}^{\mathbf{Nat}}\ \overline{3})) \\
\longrightarrow & ({}^{\mathbf{Nat}}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x})\ \overline{3}) \\
\longrightarrow & ({}^{\mathbf{Nat}}\mathsf{MS}\ \overline{3})
\end{array}
$$

first-order values are assumed to be convertible

A first attempt at polymorphism: example

$$
\begin{array}{ll}
 & ({}^{\forall\alpha.\ \alpha\to\alpha}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x}))\ \mathbf{Nat}\ \overline{3} \\
\longrightarrow & (\Lambda\alpha.\ ({}^{\alpha\to\alpha}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x}))\ \mathbf{Nat}\ \overline{3} \\
\longrightarrow & ({}^{\mathbf{Nat}\to\mathbf{Nat}}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x}))\ \overline{3} \\
\longrightarrow & (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\mathbf{Nat}}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x})\ (\mathsf{SM}^{\mathbf{Nat}}\ \mathbf{y})))\ \overline{3} \\
\longrightarrow & ({}^{\mathbf{Nat}}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x})\ (\mathsf{SM}^{\mathbf{Nat}}\ \overline{3})) \\
\longrightarrow & ({}^{\mathbf{Nat}}\mathsf{MS}\ (\lambda \mathbf{x}.\ \mathbf{x})\ \overline{3}) \\
\longrightarrow & ({}^{\mathbf{Nat}}\mathsf{MS}\ \overline{3}) \\
\longrightarrow & \overline{3}
\end{array}
$$

first-order values are assumed to be convertible

How parametricity breaks

$$(^{\forall\alpha.\ \alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))\ \mathbf{Nat}$$

How parametricity breaks

well-typed expression of type $\forall\alpha.\ \alpha \to \alpha$

$$({}^{\forall\alpha.\ \alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))\ \mathbf{Nat}$$

How parametricity breaks

well-typed expression of type $\forall\alpha.\ \alpha \to \alpha$

$$({}^{\forall\alpha.\ \alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))\ \textbf{Nat}$$

$$\longrightarrow\quad (\Lambda\alpha.\ ({}^{\alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))))\ \textbf{Nat}$$

How parametricity breaks

well-typed expression of type $\forall\alpha.\ \alpha \to \alpha$

$$({}^{\forall\alpha.\ \alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))\ \mathbf{Nat}$$

$$\longrightarrow\quad (\Lambda\alpha.\ ({}^{\alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))))\ \mathbf{Nat}$$

$$\longrightarrow\quad ({}^{\mathbf{Nat}\to\mathbf{Nat}}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))$$

How parametricity breaks

well-typed expression of type $\forall\alpha.\ \alpha \to \alpha$

$$({}^{\forall\alpha.\ \alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))\ \mathbf{Nat}$$

$$\longrightarrow\quad (\Lambda\alpha.\ ({}^{\alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))))\ \mathbf{Nat}$$

$$\longrightarrow\quad ({}^{\mathbf{Nat}\to\mathbf{Nat}}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))$$

$$\longrightarrow\quad (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\mathbf{Nat}}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))\ (\mathit{SM}^{\mathbf{Nat}}\ \mathbf{y})))$$

How parametricity breaks

well-typed expression of type $\forall\alpha.\ \alpha \to \alpha$

$$({}^{\forall\alpha.\ \alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))\ \mathbf{Nat}$$

$$\longrightarrow\quad (\Lambda\alpha.\ ({}^{\alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))))\ \mathbf{Nat}$$

$$\longrightarrow\quad ({}^{\mathbf{Nat}\to\mathbf{Nat}}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))$$

$$\longrightarrow\quad (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\mathbf{Nat}}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))\ (\mathit{SM}^{\mathbf{Nat}}\ \mathbf{y})))$$

$$\equiv\quad (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\mathbf{Nat}}\mathit{MS}\ (\lambda \mathsf{x}.\ (+\ \mathsf{x}\ \overline{1}))\ (\mathit{SM}^{\mathbf{Nat}}\ \mathbf{y})))$$

How parametricity breaks

well-typed expression of type $\forall\alpha.\ \alpha \to \alpha$

$$({}^{\forall\alpha.\ \alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))\ \mathbf{Nat}$$

$$\longrightarrow\quad (\Lambda\alpha.\ ({}^{\alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))))\ \mathbf{Nat}$$

$$\longrightarrow\quad ({}^{\mathbf{Nat}\to\mathbf{Nat}}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))$$

$$\longrightarrow\quad (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\mathbf{Nat}}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))\ (\mathit{SM}^{\mathbf{Nat}}\ \mathbf{y})))$$

$$\equiv\quad (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\mathbf{Nat}}\mathit{MS}\ (\lambda \mathsf{x}.\ (+\ \mathsf{x}\ \overline{1}))\ (\mathit{SM}^{\mathbf{Nat}}\ \mathbf{y})))$$

not the identity function!

What went wrong?

$$
\begin{array}{ll}
 & ({}^{\forall\alpha.\ \alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))) \ \mathbf{Nat} \\
\longrightarrow & (\Lambda\alpha.\ ({}^{\alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))) \ \mathbf{Nat} \\
\longrightarrow & ({}^{\mathbf{Nat}\to\mathbf{Nat}}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))) \\
\longrightarrow & (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\mathbf{Nat}}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))\ (\mathit{SM}^{\mathbf{Nat}}\ \mathbf{y}))) \\
\equiv & (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\mathbf{Nat}}\mathit{MS}\ (\lambda \mathsf{x}.\ (+\ \mathsf{x}\ \overline{1}))\ (\mathit{SM}^{\mathbf{Nat}}\ \mathbf{y})))
\end{array}
$$

not the identity function!

The problem:
Scheme is able to observe the concrete choice of type for α and behave accordingly.

Restoring parametricity

Data abstraction, revisited

Data abstraction, revisited

- Using type abstraction to enforce data abstraction is a static, compile-time approach

Data abstraction, revisited

- Using type abstraction to enforce data abstraction is a static, compile-time approach

```
c1 = {new = 0,
      inc = λx: Nat. x + 1,
      get = λx: Nat. x}

ctr1 = pack Nat, c1 as Counter
```

```
c2 = {new = 0,
      inc = λx: Int. x - 1,
      get = λx: Int. toNat(0 - x)}

ctr2 = pack Int, c2 as Counter
```

Data abstraction, revisited

- Using type abstraction to enforce data abstraction is a static, compile-time approach

```
c1 = {new = 0,
      inc = λx: Nat. x + 1,
      get = λx: Nat. x}

ctr1 = pack Nat, c1 as Counter
```

```
c2 = {new = 0,
      inc = λx: Int. x - 1,
      get = λx: Int. toNat(0 - x)}

ctr2 = pack Int, c2 as Counter
```

indistinguishable
at **compile time**

Another approach to data abstraction

Another approach to data abstraction

- Programs can create unique seals in their local scope and hand out opaque, sealed values to clients

Another approach to data abstraction

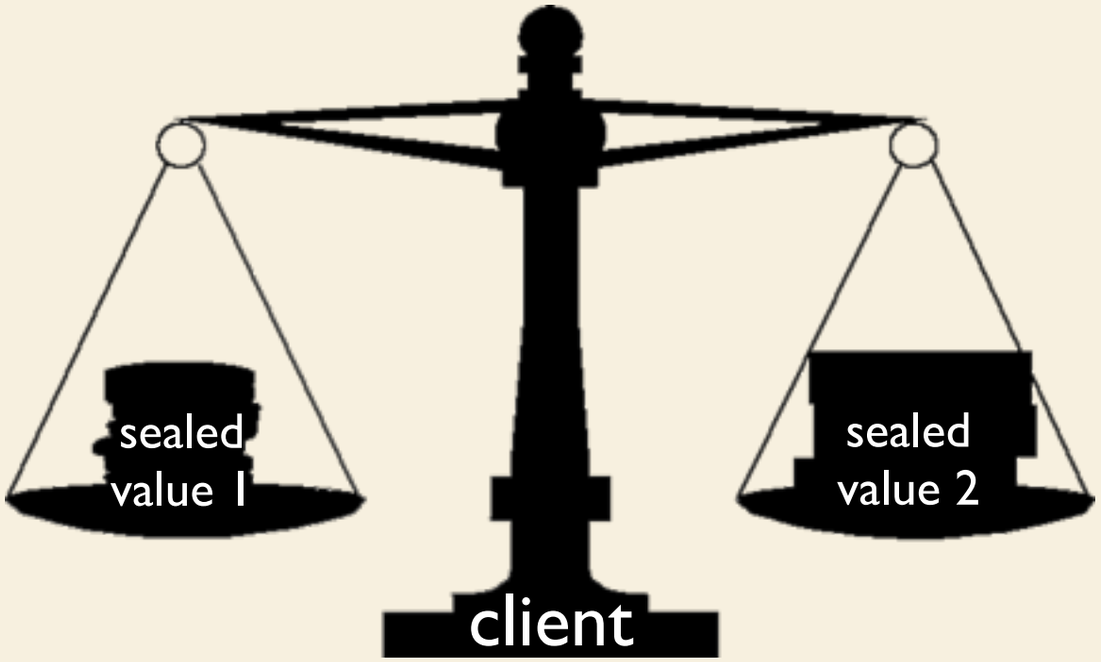
- Programs can create unique seals in their local scope and hand out opaque, sealed values to clients

```
(define create-seal) (gensym))

(define (seal-value v seal)
  (lambda (s)
    (if (eq? s seal)
        v
        (error ...))))

(define (unseal sealed-v seal)
   (sealed-v seal))
```

Another approach to data abstraction

- Programs can create unique seals in their local scope and hand out opaque, sealed values to clients

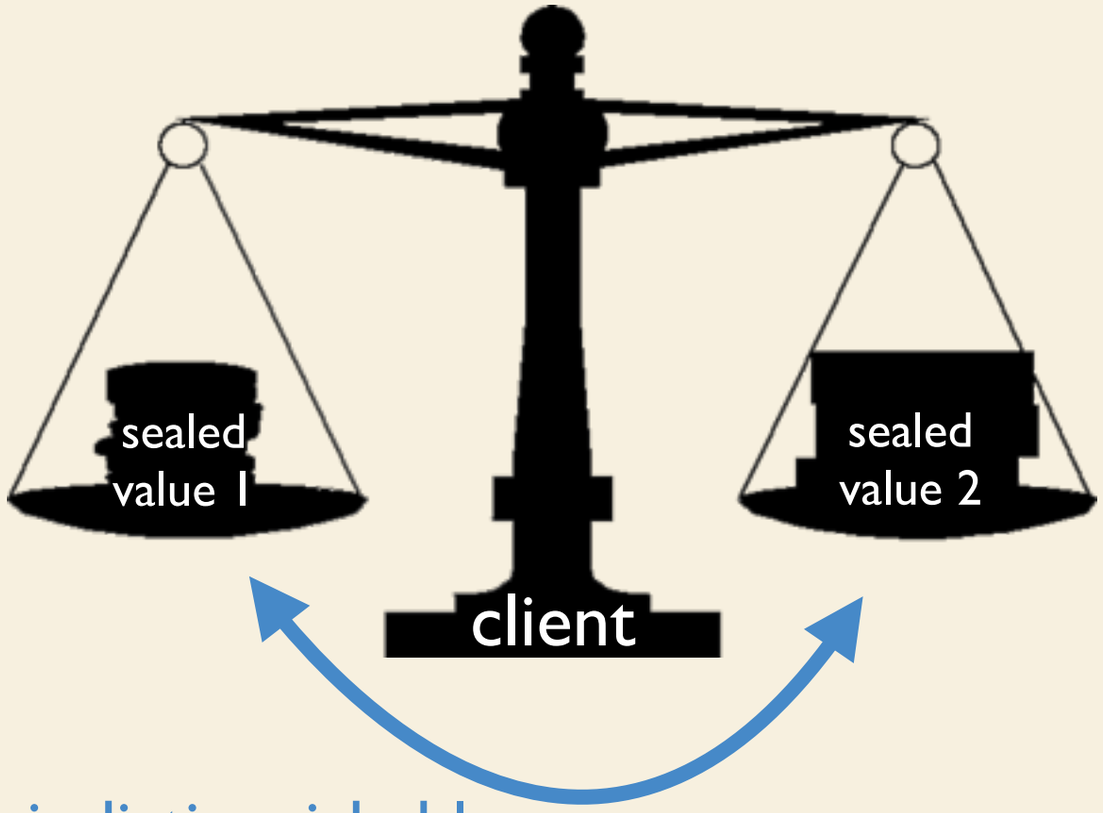
```
(define create-seal) (gensym))

(define (seal-value v seal)
  (lambda (s)
    (if (eq? s seal)
        v
        (error ...))))

(define (unseal sealed-v seal)
   (sealed-v seal))
```



Another approach to data abstraction

- Programs can create unique seals in their local scope and hand out opaque, sealed values to clients

```
(define create-seal) (gensym))

(define (seal-value v seal)
  (lambda (s)
    (if (eq? s seal)
        v
        (error ...))))

(define (unseal sealed-v seal)
   (sealed-v seal))
```



indistinguishable
at **run-time**

Updating our system to use dynamic sealing

- Operational semantics defined not just on expressions, but on configurations that include a seal store

Updating our system to use dynamic sealing

- Operational semantics defined not just on expressions, but on configurations that include a seal store

$$\psi \mid (\Lambda\alpha.\ \mathbf{e})\ \tau$$

Updating our system to use dynamic sealing

- Operational semantics defined not just on expressions, but on configurations that include a seal store

contains all seals generated during evaluation so far

$$\psi \mid (\Lambda\alpha.\ \mathbf{e})\ \tau$$

Updating our system to use dynamic sealing

- Operational semantics defined not just on expressions, but on configurations that include a seal store

contains all seals generated during evaluation so far

$$\psi \mid (\Lambda\alpha.\ \mathbf{e})\ \tau \longmapsto \psi, s \mid \mathbf{e}[\alpha := \langle s; \tau \rangle]$$

Updating our system to use dynamic sealing

- Operational semantics defined not just on expressions, but on configurations that include a seal store

contains all seals generated during evaluation so far

$$\psi \mid (\Lambda\alpha.\ \mathbf{e})\ \tau \longmapsto \psi, s \mid \mathbf{e}[\alpha := \langle s; \tau \rangle]$$

instead of regular type substitution, sealing substitution

Updating our system to use dynamic sealing

- Operational semantics defined not just on expressions, but on configurations that include a seal store

contains all seals generated during evaluation so far

$$\psi \mid (\Lambda\alpha.\ \mathbf{e})\ \tau \longmapsto \psi, s \mid \mathbf{e}[\alpha := \langle s; \tau \rangle]$$

if you think this looks *stateful*, you're right

instead of regular type substitution, sealing substitution

Back to our example...

$$({}^{\forall\alpha.\ \alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))\ \mathbf{Nat}$$

Back to our example...

well-typed expression of type $\forall\alpha.\ \alpha \to \alpha$

$$(^{\forall\alpha.\ \alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))\ \mathbf{Nat}$$

Back to our example...

well-typed expression of type $\forall\alpha.\ \alpha \to \alpha$

$$({}^{\forall\alpha.\ \alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))\ \mathbf{Nat}$$

$$\longrightarrow\quad (\Lambda\alpha.\ ({}^{\alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))))\ \mathbf{Nat}$$

Back to our example...

well-typed expression of type $\forall\alpha.\ \alpha \to \alpha$

$$({}^{\forall\alpha.\ \alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))\ \mathbf{Nat}$$

$$\longrightarrow\quad (\Lambda\alpha.\ ({}^{\alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))))\ \mathbf{Nat}$$

$$\longrightarrow\quad ({}^{\langle s;\mathbf{Nat}\rangle\to\langle s;\mathbf{Nat}\rangle}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))$$

Back to our example...

well-typed expression of type $\forall\alpha.\ \alpha \to \alpha$

$$({}^{\forall\alpha.\ \alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))\ \mathbf{Nat}$$

$$\longrightarrow\quad (\Lambda\alpha.\ ({}^{\alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))))\ \mathbf{Nat}$$

$$\longrightarrow\quad ({}^{\langle s;\mathbf{Nat}\rangle\to\langle s;\mathbf{Nat}\rangle}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))$$

$$\longrightarrow\quad (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\langle s;\mathbf{Nat}\rangle}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))\ (\mathit{SM}^{\langle s;\mathbf{Nat}\rangle}\ \mathbf{y})))$$

Back to our example...

well-typed expression of type $\forall\alpha.\ \alpha \to \alpha$

$$({}^{\forall\alpha.\ \alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))\ \mathbf{Nat}$$

$$\longrightarrow\quad (\Lambda\alpha.\ ({}^{\alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))))\ \mathbf{Nat}$$

$$\longrightarrow\quad ({}^{\langle s;\mathbf{Nat}\rangle\to\langle s;\mathbf{Nat}\rangle}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))$$

$$\longrightarrow\quad (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\langle s;\mathbf{Nat}\rangle}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))\ (\mathit{SM}^{\langle s;\mathbf{Nat}\rangle}\ \mathbf{y})))$$

opaque value

Back to our example...

well-typed expression of type $\forall\alpha.\ \alpha \to \alpha$

$$({}^{\forall\alpha.\ \alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))\ \mathbf{Nat}$$

$$\longrightarrow\quad (\Lambda\alpha.\ ({}^{\alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))))\ \mathbf{Nat}$$

$$\longrightarrow\quad ({}^{\langle s;\mathbf{Nat}\rangle\to\langle s;\mathbf{Nat}\rangle}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))$$

$$\longrightarrow\quad (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\langle s;\mathbf{Nat}\rangle}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))\ (\mathit{SM}^{\langle s;\mathbf{Nat}\rangle}\ \mathbf{y})))$$

$$\equiv\quad (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\langle s;\mathbf{Nat}\rangle}\mathit{MS}\ (\mathit{SM}^{\langle s;\mathbf{Nat}\rangle}\ \mathbf{y})))$$

opaque value

Back to our example...

well-typed expression of type $\forall\alpha.\ \alpha \to \alpha$

$$
\begin{array}{ll}
 & ({}^{\forall\alpha.\ \alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))\ \mathbf{Nat} \\
\longrightarrow & (\Lambda\alpha.\ ({}^{\alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))))\ \mathbf{Nat} \\
\longrightarrow & ({}^{\langle s;\mathbf{Nat}\rangle\to\langle s;\mathbf{Nat}\rangle}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))) \\
\longrightarrow & (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\langle s;\mathbf{Nat}\rangle}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))\ (\mathit{SM}^{\langle s;\mathbf{Nat}\rangle}\ \mathbf{y}))) \\
\equiv & (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\langle s;\mathbf{Nat}\rangle}\mathit{MS}\ (\mathit{SM}^{\langle s;\mathbf{Nat}\rangle}\ \mathbf{y}))) \\
\longrightarrow & (\lambda \mathbf{y} : \mathbf{Nat}.\ \mathbf{y})
\end{array}
$$

opaque value

Back to our example...

well-typed expression of type $\forall\alpha.\ \alpha \to \alpha$

$$
\begin{array}{ll}
 & ({}^{\forall\alpha.\ \alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))\ \mathbf{Nat} \\
\longrightarrow & (\Lambda\alpha.\ ({}^{\alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))))\ \mathbf{Nat} \\
\longrightarrow & ({}^{\langle s;\mathbf{Nat}\rangle\to\langle s;\mathbf{Nat}\rangle}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))) \\
\longrightarrow & (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\langle s;\mathbf{Nat}\rangle}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))\ (\mathit{SM}^{\langle s;\mathbf{Nat}\rangle}\ \mathbf{y}))) \\
\equiv & (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\langle s;\mathbf{Nat}\rangle}\mathit{MS}\ (\mathit{SM}^{\langle s;\mathbf{Nat}\rangle}\ \mathbf{y}))) \\
\longrightarrow & (\lambda \mathbf{y} : \mathbf{Nat}.\ \mathbf{y})
\end{array}
$$

opaque value

the identity function! :D

Another example

$$(^{\forall\alpha.\ \alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2})))\ \mathbf{Nat}\ \overline{5}$$

Another example

$$(^{\forall\alpha.\ \alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2})))\ \textbf{Nat}\ \overline{5}$$

$$\longrightarrow \quad (\Lambda\alpha.\ (^{\alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2}))))\ \textbf{Nat}\ \overline{5}$$

Another example

$$(^{\forall\alpha.\ \alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2})))\ \textbf{Nat}\ \overline{5}$$

$$\longrightarrow\quad (\Lambda\alpha.\ (^{\alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2}))))\ \textbf{Nat}\ \overline{5}$$

$$\longrightarrow\quad (^{\langle s;\textbf{Nat}\rangle\to\langle s;\textbf{Nat}\rangle}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2})))\ \overline{5}$$

Another example

$$
\begin{array}{ll}
 & ({}^{\forall\alpha.\ \alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2})))\ \mathbf{Nat}\ \overline{5} \\
\longrightarrow & (\Lambda\alpha.\ ({}^{\alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2}))))\ \mathbf{Nat}\ \overline{5} \\
\longrightarrow & ({}^{\langle s;\mathbf{Nat}\rangle\to\langle s;\mathbf{Nat}\rangle}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2})))\ \overline{5} \\
\longrightarrow & (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\langle s;\mathbf{Nat}\rangle}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2}))\ (\mathit{SM}^{\langle s;\mathbf{Nat}\rangle}\ \mathbf{y})))\ \overline{5}
\end{array}
$$

Another example

$$
\begin{array}{ll}
 & ({}^{\forall\alpha.\ \alpha\to\alpha}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2})))\ \mathbf{Nat}\ \overline{5} \\
\longrightarrow & (\Lambda\alpha.\ ({}^{\alpha\to\alpha}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2}))))\ \mathbf{Nat}\ \overline{5} \\
\longrightarrow & ({}^{\langle s;\mathbf{Nat}\rangle\to\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2})))\ \overline{5} \\
\longrightarrow & (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2}))\ (\mathsf{SM}^{\langle s;\mathbf{Nat}\rangle}\ \mathbf{y})))\ \overline{5} \\
\longrightarrow & ({}^{\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2}))\ (\mathsf{SM}^{\langle s;\mathbf{Nat}\rangle}\ \overline{5}))
\end{array}
$$

Another example

$$
\begin{array}{ll}
 & ({}^{\forall\alpha.\ \alpha\to\alpha}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2})))\ \mathbf{Nat}\ \overline{5} \\
\longrightarrow & (\Lambda\alpha.\ ({}^{\alpha\to\alpha}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2}))))\ \mathbf{Nat}\ \overline{5} \\
\longrightarrow & ({}^{\langle s;\mathbf{Nat}\rangle\to\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2})))\ \overline{5} \\
\longrightarrow & (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2}))\ (\mathsf{SM}^{\langle s;\mathbf{Nat}\rangle}\ \mathbf{y})))\ \overline{5} \\
\longrightarrow & ({}^{\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2}))\ (\mathsf{SM}^{\langle s;\mathbf{Nat}\rangle}\ \overline{5})) \\
\longrightarrow^{*} & ({}^{\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ \overline{2})
\end{array}
$$

Another example

$$(^{\forall\alpha.\ \alpha\to\alpha}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2})))\ \mathbf{Nat}\ \overline{5}$$

$$\longrightarrow\quad (\Lambda\alpha.\ (^{\alpha\to\alpha}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2}))))\ \mathbf{Nat}\ \overline{5}$$

$$\longrightarrow\quad (^{\langle s;\mathbf{Nat}\rangle\to\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2})))\ \overline{5}$$

$$\longrightarrow\quad (\lambda \mathbf{y} : \mathbf{Nat}.\ (^{\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2}))\ (\mathsf{SM}^{\langle s;\mathbf{Nat}\rangle}\ \mathbf{y})))\ \overline{5}$$

$$\longrightarrow\quad (^{\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2}))\ (\mathsf{SM}^{\langle s;\mathbf{Nat}\rangle}\ \overline{5}))$$

$$\longrightarrow^{*}\quad (^{\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ \overline{2})$$

can't unseal something
that isn't a seal

Another example

$$(^{\forall\alpha.\ \alpha\to\alpha}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2})))\ \mathbf{Nat}\ \overline{5}$$

$$\longrightarrow\quad (\Lambda\alpha.\ (^{\alpha\to\alpha}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2}))))\ \mathbf{Nat}\ \overline{5}$$

$$\longrightarrow\quad (^{\langle s;\mathbf{Nat}\rangle\to\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2})))\ \overline{5}$$

$$\longrightarrow\quad (\lambda \mathbf{y} : \mathbf{Nat}.\ (^{\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2}))\ (\mathsf{SM}^{\langle s;\mathbf{Nat}\rangle}\ \mathbf{y})))\ \overline{5}$$

$$\longrightarrow\quad (^{\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2}))\ (\mathsf{SM}^{\langle s;\mathbf{Nat}\rangle}\ \overline{5}))$$

$$\longrightarrow^{*}\quad (^{\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ \overline{2})$$

$$\longrightarrow\quad \textbf{Error}\text{: bad value}$$

can't unseal something that isn't a seal

Proving parametricity

When are two expressions indistinguishable?

- The property we really want is contextual equivalence: e_1 and e_2, when dropped into the same context, have the same observable behavior.

When are two expressions indistinguishable?

- The property we really want is contextual equivalence: e_1 and e_2, when dropped into the same context, have the same observable behavior.

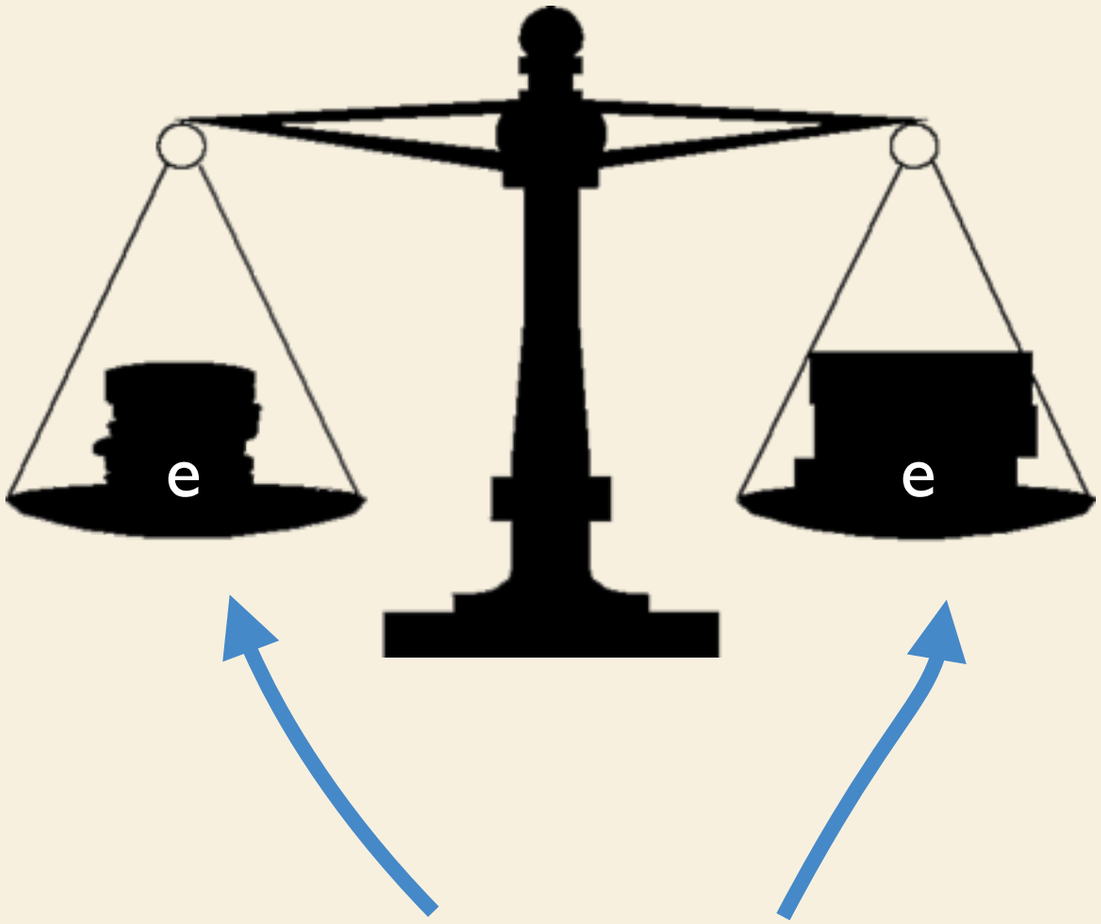
```
(if (> □ 0)
    5
    500)
```

```
(if □
    5
    500)
```

A different notion of equivalence

- Because contextual equivalence is hard to show directly, we need a different notion of equivalence.
- We'll define our own equivalence relation and show that it is sound with respect to contextual equivalence.

Reflexivity: the Fundamental Property



open expressions,
two different
closing type environments

- In order to be an *equivalence* relation, our relation has to be reflexive: every expression must be related to itself.
- But this corresponds nicely to what we mean by parametricity anyway!

What's "logical" about it?

- The relation we're defining is called a logical relation. Why?

Two values of type...	...are related if...

What's "logical" about it?

- The relation we're defining is called a logical relation. Why?

Two values of type...	...are related if...
Nat	

What's "logical" about it?

- The relation we're defining is called a logical relation. Why?

Two values of type...	...are related if...
Nat	they're equal

What's "logical" about it?

- The relation we're defining is called a logical relation. Why?

Two values of type...	...are related if...
Nat	they're equal
$\tau_1 \times \tau_2$	

What's "logical" about it?

- The relation we're defining is called a logical relation. Why?

Two values of type...	...are related if...
Nat	they're equal
$\tau_1 \times \tau_2$	their first components are related at type τ_1 **and** their second components are related at type τ_2

What's "logical" about it?

- The relation we're defining is called a logical relation. Why?

Two values of type...	...are related if...
Nat	they're equal
$\tau_1 \times \tau_2$	their first components are related at type τ_1 **and** their second components are related at type τ_2
$\tau_1 \longrightarrow \tau_2$	

What's "logical" about it?

- The relation we're defining is called a logical relation. Why?

Two values of type...	...are related if...
Nat	they're equal
$\tau_1 \times \tau_2$	their first components are related at type τ_1 **and** their second components are related at type τ_2
$\tau_1 \longrightarrow \tau_2$	**given** values related at type τ_1 they **produce** expressions related at type τ_2

What's "logical" about it?

- The relation we're defining is called a logical relation. Why?

Two values of type...	...are related if...
Nat	they're equal
$\tau_1 \times \tau_2$	their first components are related at type τ_1 **and** their second components are related at type τ_2
$\tau_1 \longrightarrow \tau_2$	**given** values related at type τ_1 they **produce** expressions related at type τ_2

- A logical relation "respects the actions of the logical operators...that correspond to the language's type constructors" (Crary, 2005)

Two values of type...	...are related if...
Nat	they're equal
$\tau_1 \times \tau_2$	their first components are related at type τ_1 **and** their second components are related at type τ_2
$\tau_1 \longrightarrow \tau_2$	**given** values related at type τ_1 they **produce** expressions related at type τ_2

A type-indexed relation

Two values of type...	...are related if...
Nat	they're equal
$\tau_1 \times \tau_2$	their first components are related at type τ_1 **and** their second components are related at type τ_2
$\tau_1 \longrightarrow \tau_2$	**given** values related at type τ_1 they **produce** expressions related at type τ_2

A type-indexed relation

Two values of type...	...are related if...
Nat	they're equal
$\tau_1 \times \tau_2$	their first components are related at type τ_1 **and** their second components are related at type τ_2
$\tau_1 \longrightarrow \tau_2$	**given** values related at type τ_1 they **produce** expressions related at type τ_2

A type-indexed relation

Two values of type...	...are related if...
Nat	they're equal
$\tau_1 \times \tau_2$	their first components are related at type τ_1 **and** their second components are related at type τ_2
$\tau_1 \longrightarrow \tau_2$	**given** values related at type τ_1 they **produce** expressions related at type τ_2
α	

A type-indexed relation

Two values of type...	...are related if...
Nat	they're equal
$\tau_1 \times \tau_2$	their first components are related at type τ_1 **and** their second components are related at type τ_2
$\tau_1 \longrightarrow \tau_2$	**given** values related at type τ_1 they **produce** expressions related at type τ_2
α	**???**

A type-indexed relation

Two values of	are related if
	ype τ_1
	type τ_2
	they **produce** expressions related at type τ_2
α	**???**

A type-indexed relation

Two values of	are related if
	type τ_1
	type τ_2
	they **produce** expressions related at type τ_2
α	**???**

$(\Lambda\alpha.\ \lambda\mathbf{x} : \alpha.\ \ldots\mathbf{x}\ldots)\ \tau_1$ $(\Lambda\alpha.\ \lambda\mathbf{x} : \alpha.\ \ldots\mathbf{x}\ldots)\ \tau_2$

A type-indexed relation

Two values of	are related if
	ype τ_1
	type τ_2
	they **produce** expressions related at type τ_2
α	**???**

$(\Lambda\alpha.\ \lambda\mathbf{x} : \alpha.\ \ldots\mathbf{x}\ldots)\ \tau_1$ $(\Lambda\alpha.\ \lambda\mathbf{x} : \alpha.\ \ldots\mathbf{x}\ldots)\ \tau_2$

related at type α iff they're in some relation **R** that relates values of type τ_1 and τ_2

A type-indexed relation

Two values of	are related if
	type τ_1
	type τ_2
	they **produce** expressions related at type τ_2
α	**???**

$(\Lambda\alpha.\ \lambda\mathbf{x} : \alpha.\ \ldots\mathbf{x}\ldots)\ \tau_1$ $(\Lambda\alpha.\ \lambda\mathbf{x} : \alpha.\ \ldots\mathbf{x}\ldots)\ \tau_2$

related at type α iff they're in some relation **R** that relates values of type τ_1 and τ_2

We parameterize the ML side of our relation with a type interpretation δ mapping type variables α to triples (τ_1, τ_2, **R**)

A type-indexed relation

Two values of	are related if
	ype τ_1
	type τ_2
	they **produce** expressions related at type τ_2
α	they belong to the relation **R** in $\delta(\alpha)$

$(\Lambda\alpha.\ \lambda\mathbf{x} : \alpha.\ \ldots\mathbf{x}\ldots)\ \tau_1$ $(\Lambda\alpha.\ \lambda\mathbf{x} : \alpha.\ \ldots\mathbf{x}\ldots)\ \tau_2$

related at type α iff they're in some relation **R** that relates values of type τ_1 and τ_2

We parameterize the ML side of our relation with a type interpretation δ mapping type variables α to triples (τ_1, τ_2, **R**)

A type-indexed relation

Two values of type...	...are related if...
Nat	they're equal
$\tau_1 \times \tau_2$	their first components are related at type τ_1 **and** their second components are related at type τ_2
$\tau_1 \longrightarrow \tau_2$	**given** values related at type τ_1 they **produce** expressions related at type τ_2
α	they belong to the relation **R** in $\delta(\alpha)$

A type-indexed relation

Two values of type...	...are related if...
Nat	they're equal
$\tau_1 \times \tau_2$	their first components are related at type τ_1 **and** their second components are related at type τ_2
$\tau_1 \longrightarrow \tau_2$	**given** values related at type τ_1 they **produce** expressions related at type τ_2
α	they belong to the relation **R** in $\delta(\alpha)$
$\forall \alpha.\ \tau$	

A type-indexed relation

Two values of type...	...are related if...
Nat	they're equal
$\tau_1 \times \tau_2$	their first components are related at type τ_1 **and** their second components are related at type τ_2
$\tau_1 \longrightarrow \tau_2$	**given** values related at type τ_1 they **produce** expressions related at type τ_2
α	they belong to the relation **R** in $\delta(\alpha)$
$\forall \alpha.\ \tau$	**???**

A type-indexed relation

Two values of type...	...are related if...
Nat	they're equal
$\tau_1 \times \tau_2$	their first components are related at type τ_1 **and** their second components are related at type τ_2
$\tau_1 \longrightarrow \tau_2$	**given** values related at type τ_1 they **produce** expressions related at type τ_2
α	they belong to the relation **R** in $\delta(\alpha)$
$\forall\alpha.\ \tau$	**given** types τ_1 and τ_2 and a relation **R** they **produce** expressions related at type τ under a δ extended with $\alpha \rightarrow (\tau_1, \tau_2, \mathbf{R})$

Two values of type...	...are related if...
Nat	they're equal
$\tau_1 \times \tau_2$	their first components are related at type τ_1 **and** their second components are related at type τ_2
$\tau_1 \to \tau_2$	**given** values related at type τ_1 they **produce** expressions related at type τ_2
α	they belong to the relation **R** in $\delta(\alpha)$
$\forall\alpha.\ \tau$	**given** types τ_1 and τ_2 and a relation **R** they **produce** expressions related at type τ under a δ extended with $\alpha \to (\tau_1, \tau_2, \mathbf{R})$

Would something like this work for Scheme?

Two values of type...	...are related if...
Nat	they're equal
$\tau_1 \times \tau_2$	their first components are related at type τ_1 **and** their second components are related at type τ_2
$\tau_1 \longrightarrow \tau_2$	**given** values related at type τ_1 they **produce** expressions related at type τ_2
α	they belong to the relation **R** in $\delta(\alpha)$
$\forall\alpha.\ \tau$	**given** types τ_1 and τ_2 and a relation **R** they **produce** expressions related at type τ under a δ extended with $\alpha \rightarrow (\tau_1, \tau_2, \mathbf{R})$

Would something like this work for Scheme?

Two values of type...	...are related if...
Nat	they're equal
$\tau_1 \times \tau_2$	their first components are related at type τ_1 **and** their second components are related at type τ_2
$\tau_1 \longrightarrow \tau_2$	**given** values related at type τ_1 they **produce** expressions related at type τ_2
α	they belong to the relation **R** in $\delta(\alpha)$
$\forall\alpha.\ \tau$	**given** types τ_1 and τ_2 and a relation **R** they **produce** expressions related at type τ under a δ extended with $\alpha \rightarrow (\tau_1, \tau_2, \mathbf{R})$

- Since Scheme only has one (static) type, a relation defined inductively on the structure of types would be ill-founded

Would something like this work for Scheme?

- Since Scheme only has one (static) type, a relation defined inductively on the structure of types would be ill-founded

Solving the ill-foundedness problem

Solving the ill-foundedness problem

- For Scheme values, index the relation by number of steps available for future computation

Solving the ill-foundedness problem

- For Scheme values, index the relation by number of steps available for future computation

Values of the syntactic form...	...are related for *j* steps if...

Solving the ill-foundedness problem

- For Scheme values, index the relation by number of steps available for future computation

Values of the syntactic form...	...are related for *j* steps if...
$\overline{n}$	

Solving the ill-foundedness problem

- For Scheme values, index the relation by number of steps available for future computation

Values of the syntactic form...	...are related for *j* steps if...
$\overline{n}$	they're equal

Solving the ill-foundedness problem

- For Scheme values, index the relation by number of steps available for future computation

Values of the syntactic form...	...are related for *j* steps if...
$\overline{n}$	they're equal
$(\text{cons } v_1\ v_2)$	

Solving the ill-foundedness problem

- For Scheme values, index the relation by number of steps available for future computation

Values of the syntactic form...	...are related for *j* steps if...
$\overline{n}$	they're equal
$(\text{cons } v_1\ v_2)$	their first components are related for *j* steps **and** their second components are related for *j* steps

Solving the ill-foundedness problem

- For Scheme values, index the relation by number of steps available for future computation

Values of the syntactic form...	...are related for *j* steps if...
$\overline{n}$	they're equal
$(\text{cons } v_1\ v_2)$	their first components are related for *j* steps **and** their second components are related for *j* steps
$(\lambda x.\ e)$	

Solving the ill-foundedness problem

- For Scheme values, index the relation by number of steps available for future computation

Values of the syntactic form...	...are related for *j* steps if...
$\overline{n}$	they're equal
$(\text{cons } v_1\ v_2)$	their first components are related for *j* steps **and** their second components are related for *j* steps
$(\lambda x.\ e)$	**???**

Examples of related Scheme values

v_1	v_2	Related (*indistinguishable*) for...
$\overline{5}$		

Examples of related Scheme values

v_1	v_2	Related (*indistinguishable*) for...
$\overline{5}$	$\overline{6}$	

Examples of related Scheme values

v_1	v_2	Related (*indistinguishable*) for...
$\overline{5}$	$\overline{6}$	0 steps

Examples of related Scheme values

v_1	v_2	Related (*indistinguishable*) for...
$\overline{5}$	$\overline{6}$	0 steps
$(\lambda \mathbf{x}.\ \overline{5})$	$(\lambda \mathbf{x}.\ \overline{6})$	

Examples of related Scheme values

v_1	v_2	Related (*indistinguishable*) for...
$\overline{5}$	$\overline{6}$	0 steps
$(\lambda \mathsf{x}.\ \overline{5})$	$(\lambda \mathsf{x}.\ \overline{6})$	

$$(\lambda \mathsf{x}.\ \overline{5})\ \overline{1} \longmapsto \overline{5}[\mathsf{x} := \overline{1}] \longmapsto \overline{5}$$
$$(\lambda \mathsf{x}.\ \overline{6})\ \overline{1} \longmapsto \overline{6}[\mathsf{x} := \overline{1}] \longmapsto \overline{6}$$

Substitution uses up 1 step

Examples of related Scheme values

v_1	v_2	Related (*indistinguishable*) for...
$\overline{5}$	$\overline{6}$	0 steps
$(\lambda \mathsf{x}.\ \overline{5})$	$(\lambda \mathsf{x}.\ \overline{6})$	1 step

$$(\lambda \mathsf{x}.\ \overline{5})\ \overline{1} \longmapsto \overline{5}[\mathsf{x} := \overline{1}] \longmapsto \overline{5}$$
$$(\lambda \mathsf{x}.\ \overline{6})\ \overline{1} \longmapsto \overline{6}[\mathsf{x} := \overline{1}] \longmapsto \overline{6}$$

Substitution uses up 1 step

Examples of related Scheme values

v_1	v_2	Related (*indistinguishable*) for...
$\overline{5}$	$\overline{6}$	0 steps
$(\lambda \mathbf{x}.\ \overline{5})$	$(\lambda \mathbf{x}.\ \overline{6})$	1 step

Examples of related Scheme values

v_1	v_2	Related (*indistinguishable*) for...
$\overline{5}$	$\overline{6}$	0 steps
$(\lambda \mathbf{x}.\ \overline{5})$	$(\lambda \mathbf{x}.\ \overline{6})$	1 step
$(\lambda \mathbf{y}.\ (\lambda \mathbf{x}.\ \overline{5}))$		

Examples of related Scheme values

v_1	v_2	Related (*indistinguishable*) for...
$\overline{5}$	$\overline{6}$	0 steps
$(\lambda \mathbf{x}.\ \overline{5})$	$(\lambda \mathbf{x}.\ \overline{6})$	1 step
$(\lambda \mathbf{y}.\ (\lambda \mathbf{x}.\ \overline{5}))$	$(\lambda \mathbf{y}.\ (\lambda \mathbf{x}.\ \overline{6}))$	

Examples of related Scheme values

v_1	v_2	Related (*indistinguishable*) for...
$\overline{5}$	$\overline{6}$	0 steps
$(\lambda \mathbf{x}.\ \overline{5})$	$(\lambda \mathbf{x}.\ \overline{6})$	1 step
$(\lambda \mathbf{y}.\ (\lambda \mathbf{x}.\ \overline{5}))$	$(\lambda \mathbf{y}.\ (\lambda \mathbf{x}.\ \overline{6}))$	

$$(\lambda \mathbf{y}.\ (\lambda \mathbf{x}.\ \overline{5}))\ \overline{1} \longmapsto (\lambda \mathbf{x}.\ \overline{5})[\mathbf{y} := \overline{1}] \longmapsto (\lambda \mathbf{x}.\ \overline{5})$$
$$(\lambda \mathbf{y}.\ (\lambda \mathbf{x}.\ \overline{6}))\ \overline{1} \longmapsto (\lambda \mathbf{x}.\ \overline{6})[\mathbf{y} := \overline{1}] \longmapsto (\lambda \mathbf{x}.\ \overline{6})$$

Examples of related Scheme values

v_1	v_2	Related (*indistinguishable*) for...
$\overline{5}$	$\overline{6}$	0 steps
$(\lambda \mathsf{x}.\ \overline{5})$	$(\lambda \mathsf{x}.\ \overline{6})$	1 step
$(\lambda \mathsf{y}.\ (\lambda \mathsf{x}.\ \overline{5}))$	$(\lambda \mathsf{y}.\ (\lambda \mathsf{x}.\ \overline{6}))$	2 steps

$$(\lambda \mathsf{y}.\ (\lambda \mathsf{x}.\ \overline{5}))\ \overline{1} \longmapsto (\lambda \mathsf{x}.\ \overline{5})[\mathsf{y} := \overline{1}] \longmapsto (\lambda \mathsf{x}.\ \overline{5})$$
$$(\lambda \mathsf{y}.\ (\lambda \mathsf{x}.\ \overline{6}))\ \overline{1} \longmapsto (\lambda \mathsf{x}.\ \overline{6})[\mathsf{y} := \overline{1}] \longmapsto (\lambda \mathsf{x}.\ \overline{6})$$

Examples of related Scheme values

v_1	v_2	Related (*indistinguishable*) for...
$\overline{5}$	$\overline{6}$	0 steps
$(\lambda \mathbf{x}.\ \overline{5})$	$(\lambda \mathbf{x}.\ \overline{6})$	1 step
$(\lambda \mathbf{y}.\ (\lambda \mathbf{x}.\ \overline{5}))$	$(\lambda \mathbf{y}.\ (\lambda \mathbf{x}.\ \overline{6}))$	2 steps

Examples of related Scheme values

v_1	v_2	Related (*indistinguishable*) for...
$\overline{5}$	$\overline{6}$	0 steps
$(\lambda \mathbf{x}.\ \overline{5})$	$(\lambda \mathbf{x}.\ \overline{6})$	1 step
$(\lambda \mathbf{y}.\ (\lambda \mathbf{x}.\ \overline{5}))$	$(\lambda \mathbf{y}.\ (\lambda \mathbf{x}.\ \overline{6}))$	2 steps

- Intuitively, wrapping layers of λ around values makes them indistinguishable for 1 more step

A step-indexed relation

Two values of the syntactic form...	...are related for *j* steps if...
$\overline{n}$	they're equal
$(\text{cons } \mathsf{v}_1\ \mathsf{v}_2)$	their first components are related for *j* steps **and** their second components are related for *j* steps
$(\lambda \mathsf{x}.\ \mathsf{e})$	**???**

A step-indexed relation

Two values of the syntactic form...	...are related for *j* steps if...
$\overline{n}$	they're equal
$(\text{cons } v_1\ v_2)$	their first components are related for *j* steps **and** their second components are related for *j* steps
$(\lambda x.\ e)$	**given** values related for $i < j$ steps they **produce** expressions related for *i* steps

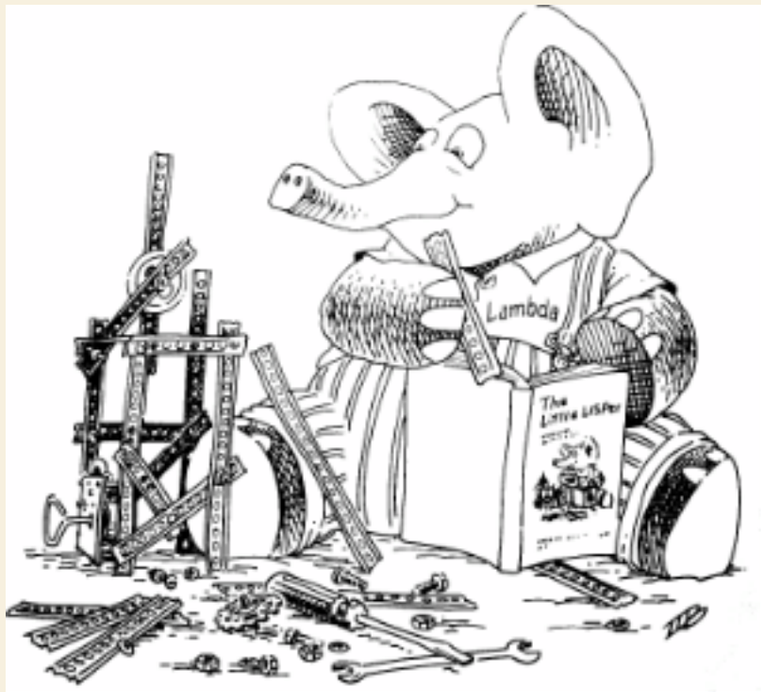
The little step-indexer

The little step-indexer

- Chapter 9 of *The Little Schemer* gives examples of functions **length**$_0$, **length**$_{\leq 1}$, **length**$_{\leq 2}$, and so on

The little step-indexer

- Chapter 9 of *The Little Schemer* gives examples of functions **length**$_0$, **length**$_{\leq 1}$, **length**$_{\leq 2}$, and so on
- **length**$_{\leq j}$ takes a list and returns the length of that list, as long as that length is $\leq j$; otherwise, **length**$_{\leq j}$ goes into an infinite loop



The little step-indexer

- Chapter 9 of *The Little Schemer* gives examples of functions **length**$_0$, **length**$_{\leq 1}$, **length**$_{\leq 2}$, and so on
- **length**$_{\leq j}$ takes a list and returns the length of that list, as long as that length is $\leq j$; otherwise, **length**$_{\leq j}$ goes into an infinite loop

- Think of the subscript $\leq j$ as a behavioral contract guaranteeing that **length**$_{\leq j}$ belongs to a certain type for up to j steps of execution

The little step-indexer

- Chapter 9 of *The Little Schemer* gives examples of functions **length**$_0$, **length**$_{\leq 1}$, **length**$_{\leq 2}$, and so on
- **length**$_{\leq j}$ takes a list and returns the length of that list, as long as that length is $\leq j$; otherwise, **length**$_{\leq j}$ goes into an infinite loop

- Think of the subscript $\leq j$ as a behavioral contract guaranteeing that **length**$_{\leq j}$ belongs to a certain type for up to j steps of execution
- This is exactly the intuition behind the step-indexed model of recursive types (Appel & McAllester, 2001)

A step-indexed relation

Two values of the syntactic form...	...are related for *j* steps if...
$\overline{n}$	they're equal
$(\text{cons } v_1\ v_2)$	their first components are related for *j* steps **and** their second components are related for *j* steps
$(\lambda x.\ e)$	**given** values related for $i < j$ steps they **produce** expressions related for *i* steps

A step-indexed relation

Two values of the syntactic form...	...are related for *j* steps if...
$\overline{n}$	they're equal
$(\text{cons } v_1\ v_2)$	their first components are related for *j* steps **and** their second components are related for *j* steps
$(\lambda x.\ e)$	**given** values related for *i* < *j* steps they **produce** expressions related for *i* steps
$(\mathcal{SM}^{\langle s;\tau\rangle}\ \mathbf{v})$	

A step-indexed relation

Two values of the syntactic form...	...are related for *j* steps if...
$\overline{n}$	they're equal
$(\text{cons } \mathsf{v}_1\ \mathsf{v}_2)$	their first components are related for *j* steps **and** their second components are related for *j* steps
$(\lambda \mathsf{x}.\ \mathsf{e})$	**given** values related for *i* < *j* steps they **produce** expressions related for *i* steps
$(\mathcal{SM}^{\langle s;\tau\rangle}\ \mathbf{v})$	**???**

A step-indexed relation

Two values of the syntactic form...	...are related for *j* steps if...
$\overline{n}$	they're equal
$(\text{cons } v_1\ v_2)$	their first components are related for *j* steps **and** their second components are related for *j* steps
$(\lambda x.\ e)$	**given** values related for *i* < *j* steps they **produce** expressions related for *i* steps
$(SM^{\langle s;\tau\rangle}\ \mathbf{v})$	the inner ML expressions are related for *j-1* steps

A step-indexed relation

Two values of the syntactic form...	...are related for *j* steps if...
$\overline{n}$	they're equal
$(\text{cons } v_1\ v_2)$	their first components are related for *j* steps **and** their second components are related for *j* steps
$(\lambda x.\ e)$	**given** values related for *i* < *j* steps they **produce** expressions related for *i* steps
$(SM^{\langle s;\tau\rangle}\ \mathbf{v})$	the inner ML expressions are related for *j-1* steps

step indices "leak" back into the ML relation

$(SM^{\langle s;\tau\rangle}\ \mathbf{v})$	the inner ML expressions are related for *j-1* steps

But wait!

$(\mathcal{SM}^{\langle s;\tau\rangle}\ \mathbf{v})$	the inner ML expressions are related for *j-1* steps

But wait!

$(\mathcal{M}^{\langle s;\tau\rangle}\ \mathbf{v})$	the inner ML expressions are related for *j-1* steps

$$(\mathcal{M}^{\langle s_1;\tau_1\rangle}\ \mathbf{v}_1) \qquad (\mathcal{M}^{\langle s_2;\tau_2\rangle}\ \mathbf{v}_2)$$

At what *type* are $\mathbf{v}_1$ and $\mathbf{v}_2$ related?

But wait!

$(SM^{\langle s;\tau\rangle}\ \mathbf{v})$	the inner ML expressions are related for *j-1* steps

$(SM^{\langle s_1;\tau_1\rangle}\ \mathbf{v}_1)$ $(SM^{\langle s_2;\tau_2\rangle}\ \mathbf{v}_2)$

At what *type* are $\mathbf{v_1}$ and $\mathbf{v_2}$ related?

- The type of these sealed values was originally a type variable...

But wait!

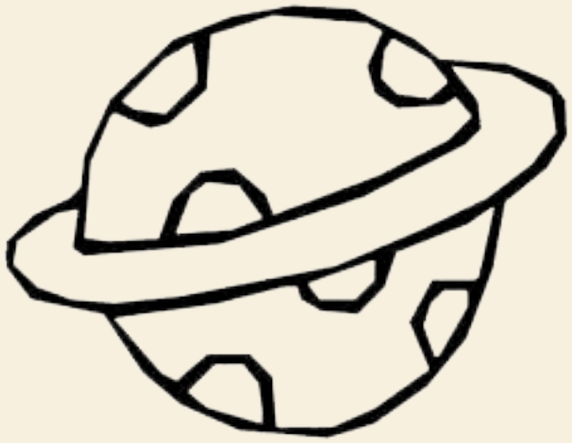
$(SM^{\langle s;\tau\rangle}\ \mathbf{v})$	the inner ML expressions are related for *j-1* steps

$$(SM^{\langle s_1;\tau_1\rangle}\ \mathbf{v}_1) \qquad (SM^{\langle s_2;\tau_2\rangle}\ \mathbf{v}_2)$$

At what *type* are $\mathbf{v_1}$ and $\mathbf{v_2}$ related?

- The type of these sealed values was originally a type variable...
- We need a dynamic counterpart to δ

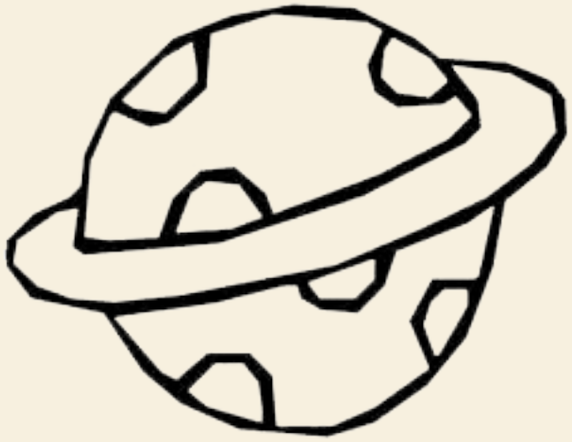
Possible worlds



- An idea from modal logic (Kripke, 1963)
- Useful for reasoning about properties that only hold under certain conditions

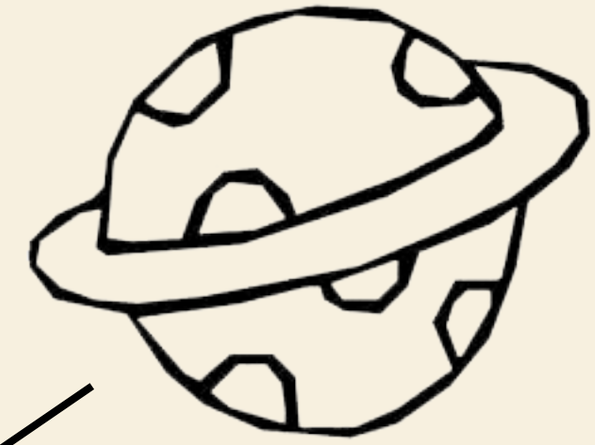
What's in a world?

"Meanwhile, in the world where
$\mathbf{e_1}$ *and* **$\mathbf{e_2}$** *are related..."*



What's in a world?

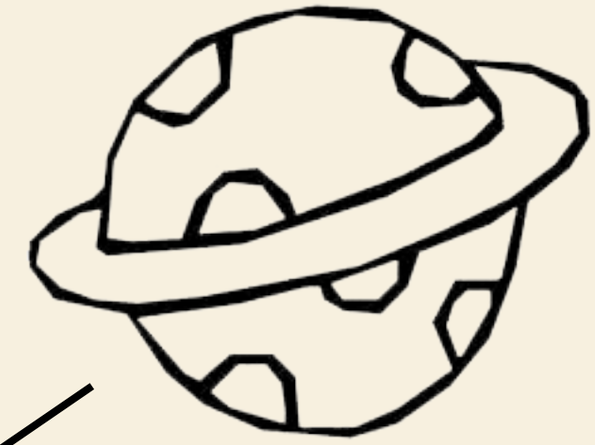
"Meanwhile, in the world where **e_1** *and* **e_2** *are related..."*



seals s_1 generated during evaluation of **e_1**

What's in a world?

"Meanwhile, in the world where **e_1** *and* **e_2** *are related..."*

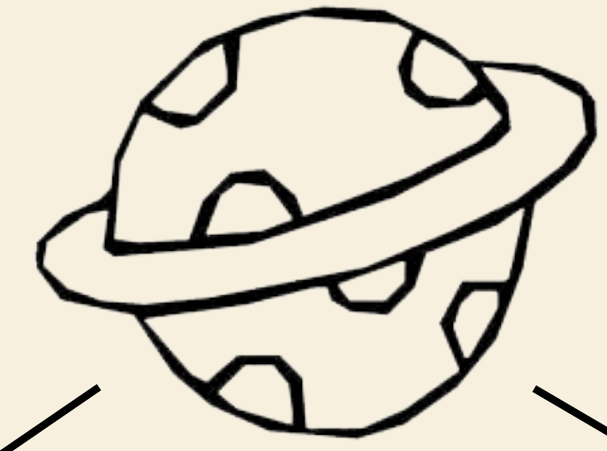


seals s_1 generated during evaluation of **e_1**

seals s_2 generated during evaluation of **e_2**

What's in a world?

"Meanwhile, in the world where **e_1** *and* **e_2** *are related..."*



seals s_1 generated during evaluation of **e_1**

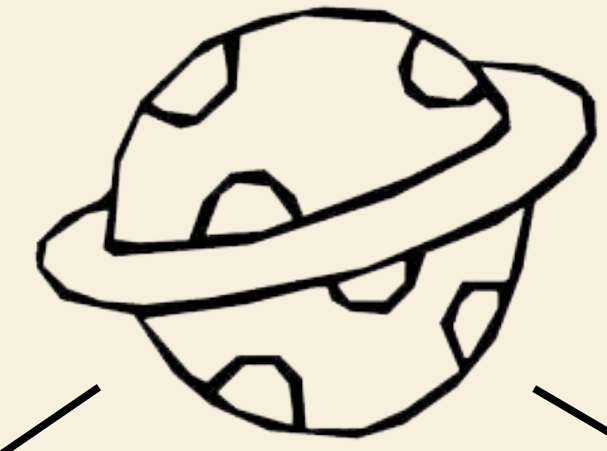
seals s_2 generated during evaluation of **e_2**

mappings

$\alpha \rightarrow (s_1, s_2)$

What's in a world?

"Meanwhile, in the world where **e_1** *and* **e_2** *are related..."*



seals s_1 generated during evaluation of **e_1**

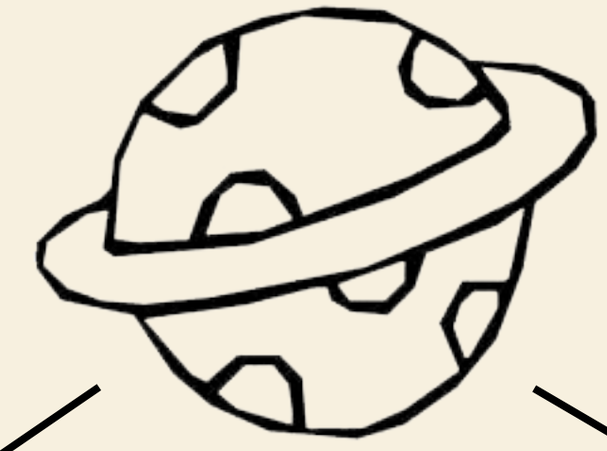
seals s_2 generated during evaluation of **e_2**

mappings $\alpha \rightarrow (s_1, s_2)$

mappings $\alpha \rightarrow (\tau_1, \tau_2, \mathbf{R})$

What's in a world?

"Meanwhile, in the world where **e_1** *and* **e_2** *are related..."*



seals s_1 generated during evaluation of **e_1**

seals s_2 generated during evaluation of **e_2**

mappings $\alpha \rightarrow (s_1, s_2)$

mappings $\alpha \rightarrow (\tau_1, \tau_2, \mathbf{R})$

- Worlds capture the relationship between static type variables and dynamic seals

Relatedness in a world

$(\mathcal{SM}^{\langle s_1;\tau_1\rangle}\ \mathbf{v}_1)$ $(\mathcal{SM}^{\langle s_2;\tau_2\rangle}\ \mathbf{v}_2)$

At what *type* are $\mathbf{v_1}$ and $\mathbf{v_2}$ related?

Relatedness in a world

$(\mathcal{SM}^{\langle s_1;\tau_1\rangle}\ \mathbf{v}_1)$ $(\mathcal{SM}^{\langle s_2;\tau_2\rangle}\ \mathbf{v}_2)$

At what *type* are $\mathbf{v_1}$ and $\mathbf{v_2}$ related?

- The answer: $\mathbf{v_1}$ and $\mathbf{v_2}$ must belong to a relation **R** that relates values of type τ_1 and τ_2

Relatedness in a world

$(\mathcal{SM}^{\langle s_1;\tau_1\rangle}\ \mathbf{v}_1)$ $(\mathcal{SM}^{\langle s_2;\tau_2\rangle}\ \mathbf{v}_2)$

At what *type* are $\mathbf{v_1}$ and $\mathbf{v_2}$ related?

- The answer: $\mathbf{v_1}$ and $\mathbf{v_2}$ must belong to a relation **R** that relates values of type τ_1 and τ_2
- We can find **R** in the current world

A possible-worlds model

A possible-worlds model

- Expressions are now related at a type, for a given number of steps, and in a world

A possible-worlds model

- Expressions are now related at a type, for a given number of steps, and in a world
- Whenever we do type application, we extend the current world with new seals s_1 and s_2 and new bindings for α

A possible-worlds model

- Expressions are now related at a type, for a given number of steps, and in a world
- Whenever we do type application, we extend the current world with new seals s_1 and s_2 and new bindings for α
- Whenever we need to determine relatedness of sealed values, we consult the current world to find the **R** that would relate them

A possible-worlds model

- Expressions are now related at a type, for a given number of steps, and in a world
- Whenever we do type application, we extend the current world with new seals s_1 and s_2 and new bindings for α
- Whenever we need to determine relatedness of sealed values, we consult the current world to find the **R** that would relate them
- Upshot of all this: now we can prove parametricity!

Sage advice

Sage advice


@sstrickl
Stevie Strickland
When in doubt, add another environment to your relation. #typesystemprotips
30 Mar via TweetDeck
Unfavorite
Retweet
Reply

The Fundamental Property / Parametricity

The Fundamental Property / Parametricity

- The bridge lemma:

The Fundamental Property / Parametricity

- The bridge lemma:

1. For all e_1 and e_2,
   if $(j, w, \mathsf{e}_1, \mathsf{e}_2) \in \mathcal{V}_S$
   then $(j, w, ({}^{\delta_1(\tau)}\mathsf{MS}\ \mathsf{e}_1), ({}^{\delta_2(\tau)}\mathsf{MS}\ \mathsf{e}_2)) \in \mathcal{V}_M[\![\tau]\!]\delta.$

The Fundamental Property / Parametricity

- The bridge lemma:

  1. For all e_1 and e_2,
     if $(j, w, \mathsf{e}_1, \mathsf{e}_2) \in \mathcal{V}_S$
     then $(j, w, ({}^{\delta_1(\tau)}\mathsf{MS}\ \mathsf{e}_1), ({}^{\delta_2(\tau)}\mathsf{MS}\ \mathsf{e}_2)) \in \mathcal{V}_M[\![\tau]\!]\delta.$

carries relatedness between languages

The Fundamental Property / Parametricity

- The bridge lemma:

carries relatedness between languages

1. For all $\mathbf{e}_1$ and $\mathbf{e}_2$,
   if $(j, w, \mathbf{e}_1, \mathbf{e}_2) \in \mathcal{V}_S$
   then $(j, w, ({}^{\delta_1(\tau)}\mathcal{MS}\ \mathbf{e}_1), ({}^{\delta_2(\tau)}\mathcal{MS}\ \mathbf{e}_2)) \in \mathcal{V}_M[\![\tau]\!]\delta.$

2. For all $\mathbf{e}_1$ and $\mathbf{e}_2$,
   if $(j, w, \mathbf{e}_1, \mathbf{e}_2) \in \mathcal{V}_M[\![\tau]\!]\delta$
   then $(j, w, (\mathcal{SM}^{\delta_1(\tau)}\ \mathbf{e}_1), (\mathcal{SM}^{\delta_2(\tau)}\ \mathbf{e}_2)) \in \mathcal{V}_S.$

The Fundamental Property / Parametricity

- The bridge lemma:

carries relatedness between languages

1. For all e_1 and e_2,
   if $(j, w, \mathsf{e}_1, \mathsf{e}_2) \in \mathcal{V}_S$
   then $(j, w, ({}^{\delta_1(\tau)}\mathcal{MS}\ \mathsf{e}_1), ({}^{\delta_2(\tau)}\mathcal{MS}\ \mathsf{e}_2)) \in \mathcal{V}_M[\![\tau]\!]\delta.$

2. For all $\mathbf{e}_1$ and $\mathbf{e}_2$,
   if $(j, w, \mathbf{e}_1, \mathbf{e}_2) \in \mathcal{V}_M[\![\tau]\!]\delta$
   then $(j, w, (\mathcal{SM}^{\delta_1(\tau)}\ \mathbf{e}_1), (\mathcal{SM}^{\delta_2(\tau)}\ \mathbf{e}_2)) \in \mathcal{V}_S.$

- From there we can show the Fundamental Property:

The Fundamental Property / Parametricity

- The bridge lemma:

carries relatedness between languages

1. For all $\mathbf{e}_1$ and $\mathbf{e}_2$,
   if $(j, w, \mathbf{e}_1, \mathbf{e}_2) \in \mathcal{V}_S$
   then $(j, w, ({}^{\delta_1(\tau)}\mathcal{MS}\ \mathbf{e}_1), ({}^{\delta_2(\tau)}\mathcal{MS}\ \mathbf{e}_2)) \in \mathcal{V}_M[\![\tau]\!]\delta.$

2. For all $\mathbf{e}_1$ and $\mathbf{e}_2$,
   if $(j, w, \mathbf{e}_1, \mathbf{e}_2) \in \mathcal{V}_M[\![\tau]\!]\delta$
   then $(j, w, (\mathcal{SM}^{\delta_1(\tau)}\ \mathbf{e}_1), (\mathcal{SM}^{\delta_2(\tau)}\ \mathbf{e}_2)) \in \mathcal{V}_S.$

- From there we can show the Fundamental Property:

1. If $\Delta; \Gamma \vdash_M \mathbf{e} : \tau$, then $\Delta; \Gamma \vdash_M \mathbf{e} \lesssim_M \mathbf{e} : \tau$.

The Fundamental Property / Parametricity

- The bridge lemma:

carries relatedness between languages

1. For all e_1 and e_2,
   if $(j, w, \mathsf{e}_1, \mathsf{e}_2) \in \mathcal{V}_S$
   then $(j, w, ({}^{\delta_1(\tau)}\mathsf{MS}\ \mathsf{e}_1), ({}^{\delta_2(\tau)}\mathsf{MS}\ \mathsf{e}_2)) \in \mathcal{V}_M[\![\tau]\!]\delta.$

2. For all $\mathbf{e}_1$ and $\mathbf{e}_2$,
   if $(j, w, \mathbf{e}_1, \mathbf{e}_2) \in \mathcal{V}_M[\![\tau]\!]\delta$
   then $(j, w, (\mathsf{SM}^{\delta_1(\tau)}\ \mathbf{e}_1), (\mathsf{SM}^{\delta_2(\tau)}\ \mathbf{e}_2)) \in \mathcal{V}_S.$

- From there we can show the Fundamental Property:

1. If $\Delta; \Gamma \vdash_M \mathbf{e} : \tau$, then $\Delta; \Gamma \vdash_M \mathbf{e} \lesssim_M \mathbf{e} : \tau$.

2. If $\Delta; \Gamma \vdash_S \mathsf{e} : \mathbf{TST}$, then $\Delta; \Gamma \vdash_S \mathsf{e} \lesssim_S \mathsf{e} : \mathbf{TST}$.

Parametric contracted Scheme terms

Parametric contracted Scheme terms

- One way to enforce a contract τ on a Scheme expression is by exporting it into ML at the type τ and then importing it back into Scheme...

Parametric contracted Scheme terms

- One way to enforce a contract τ on a Scheme expression is by exporting it into ML at the type τ and then importing it back into Scheme...

$$e^{\tau} = (SM^{\tau}\ ({}^{\tau}MS\ e))$$

Parametric contracted Scheme terms

- One way to enforce a contract τ on a Scheme expression is by exporting it into ML at the type τ and then importing it back into Scheme...

$$e^{\tau} = (SM^{\tau}\ ({}^{\tau}MS\ e))$$

- ...so we can leverage our parametricity result to immediately show that contracted Scheme terms behave parametrically too

Conclusion

The three points I want you to remember

The three points I want you to remember

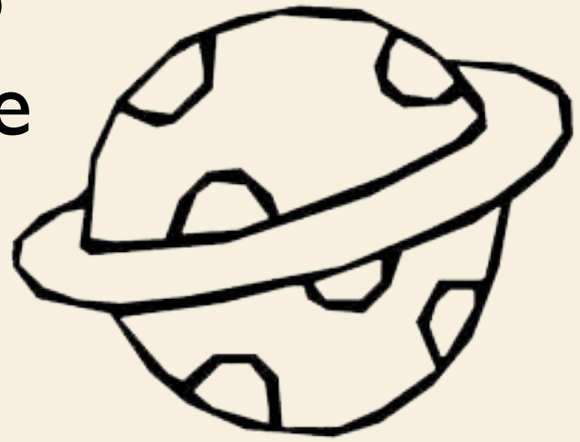
- Aside from giving us free theorems, **parametricity** makes existential-style data abstraction possible.

The three points I want you to remember

- Aside from giving us free theorems, **parametricity** makes existential-style data abstraction possible.
- Parametricity breaks when we incorporate dynamically typed code into otherwise statically typed programs, but we can restore it using **dynamic seal generation**.

The three points I want you to remember

- Aside from giving us free theorems, **parametricity** makes existential-style data abstraction possible.
- Parametricity breaks when we incorporate dynamically typed code into otherwise statically typed programs, but we can restore it using **dynamic seal generation**.
- Seal generation is a stateful notion akin to dynamic memory allocation, so we can use **possible worlds** to reason about the semantics of seals in order to prove parametricity.



Thanks!

Email: lkuper@cs.indiana.edu
Web: www.cs.indiana.edu/~lkuper
Research group: lambda.soic.indiana.edu

Detailed non-parametricity example

$$
\begin{array}{ll}
 & ({}^{\forall\alpha.\ \alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))\ \mathbf{Nat}\ \overline{5} \\
\longrightarrow & (\Lambda\alpha.\ ({}^{\alpha\to\alpha}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))))\ \mathbf{Nat}\ \overline{5} \\
\longrightarrow & ({}^{\mathbf{Nat}\to\mathbf{Nat}}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))\ \overline{5} \\
\longrightarrow & (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\mathbf{Nat}}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))\ (\mathit{SM}^{\mathbf{Nat}}\ \mathbf{y})))\ \overline{5} \\
\longrightarrow & ({}^{\mathbf{Nat}}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))\ (\mathit{SM}^{\mathbf{Nat}}\ \overline{5})) \\
\longrightarrow & ({}^{\mathbf{Nat}}\mathit{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))\ \overline{5}) \\
\longrightarrow & ({}^{\mathbf{Nat}}\mathit{MS}\ (\mathsf{if0}\ (\mathsf{nat?}\ \overline{5})\ (+\ \overline{5}\ \overline{1})\ \overline{5})) \\
\longrightarrow & ({}^{\mathbf{Nat}}\mathit{MS}\ (\mathsf{if0}\ \overline{0}\ (+\ \overline{5}\ \overline{1})\ \overline{5})) \\
\longrightarrow & ({}^{\mathbf{Nat}}\mathit{MS}\ (+\ \overline{5}\ \overline{1})) \\
\longrightarrow & ({}^{\mathbf{Nat}}\mathit{MS}\ \overline{6}) \\
\longrightarrow & \overline{6}
\end{array}
$$

Detailed dynamic sealing example

$$
\begin{array}{ll}
 & ({}^{\forall\alpha.\ \alpha\to\alpha}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))\ \mathbf{Nat}\ \overline{5} \\
\longrightarrow & (\Lambda\alpha.\ ({}^{\alpha\to\alpha}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))))\ \mathbf{Nat}\ \overline{5} \\
\longrightarrow & ({}^{\langle s;\mathbf{Nat}\rangle\to\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x})))\ \overline{5} \\
\longrightarrow & (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))\ (\mathsf{SM}^{\langle s;\mathbf{Nat}\rangle}\ \mathbf{y})))\ \overline{5} \\
\longrightarrow & ({}^{\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))\ (\mathsf{SM}^{\langle s;\mathbf{Nat}\rangle}\ \overline{5})) \\
\longrightarrow & ({}^{\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \mathsf{x}))\ (\mathsf{SM}^{\langle s;\mathbf{Nat}\rangle}\ \overline{5})) \\
\longrightarrow & ({}^{\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ (\mathsf{if0}\ (\mathsf{nat?}\ (\mathsf{SM}^{\langle s;\mathbf{Nat}\rangle}\ \overline{5}))\ (+\ (\mathsf{SM}^{\langle s;\mathbf{Nat}\rangle}\ \overline{5})\ \overline{1})\ (\mathsf{SM}^{\langle s;\mathbf{Nat}\rangle}\ \overline{5}))) \\
\longrightarrow & ({}^{\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ (\mathsf{if0}\ \overline{1}\ (+\ (\mathsf{SM}^{\langle s;\mathbf{Nat}\rangle}\ \overline{5})\ \overline{1})\ (\mathsf{SM}^{\langle s;\mathbf{Nat}\rangle}\ \overline{5}))) \\
\longrightarrow & ({}^{\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ (\mathsf{SM}^{\langle s;\mathbf{Nat}\rangle}\ \overline{5})) \\
\longrightarrow & \overline{5}
\end{array}
$$

Another detailed dynamic sealing example

$$
\begin{array}{ll}
 & ({}^{\forall\alpha.\ \alpha\to\alpha}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2})))\ \mathbf{Nat}\ \overline{5} \\
\longrightarrow & (\Lambda\alpha.\ ({}^{\alpha\to\alpha}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2}))))\ \mathbf{Nat}\ \overline{5} \\
\longrightarrow & ({}^{\langle s;\mathbf{Nat}\rangle\to\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2})))\ \overline{5} \\
\longrightarrow & (\lambda \mathbf{y} : \mathbf{Nat}.\ ({}^{\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2}))\ (\mathsf{SM}^{\langle s;\mathbf{Nat}\rangle}\ \mathbf{y})))\ \overline{5} \\
\longrightarrow & ({}^{\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2}))\ (\mathsf{SM}^{\langle s;\mathbf{Nat}\rangle}\ \overline{5})) \\
\longrightarrow & ({}^{\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ (\lambda \mathsf{x}.\ (\mathsf{if0}\ (\mathsf{nat?}\ \mathsf{x})\ (+\ \mathsf{x}\ \overline{1})\ \overline{2}))\ (\mathsf{SM}^{\langle s;\mathbf{Nat}\rangle}\ \overline{5})) \\
\longrightarrow & ({}^{\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ (\mathsf{if0}\ (\mathsf{nat?}\ (\mathsf{SM}^{\langle s;\mathbf{Nat}\rangle}\ \overline{5}))\ (+\ (\mathsf{SM}^{\langle s;\mathbf{Nat}\rangle}\ \overline{5})\ \overline{1})\ \overline{2})) \\
\longrightarrow & ({}^{\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ (\mathsf{if0}\ \overline{1}\ (+\ (\mathsf{SM}^{\langle s;\mathbf{Nat}\rangle}\ \overline{5})\ \overline{1})\ \overline{2})) \\
\longrightarrow & ({}^{\langle s;\mathbf{Nat}\rangle}\mathsf{MS}\ \overline{2}) \\
\longrightarrow & \textbf{Error}\text{: bad value}
\end{array}
$$