Free Theorems — Foundations

Janis Voigtländer

University of Bonn

October 18th, 2010

Using a Free Theorem [Wadler 1989]

For every

$$\texttt{get} :: [\alpha] \to [\alpha]$$

we have

$$\texttt{map}\ f\ (\texttt{get}\ l) \quad = \quad \texttt{get}\ (\texttt{map}\ f\ l)$$

for arbitrary f and l, where

$$\begin{aligned}
&\texttt{map} :: (\alpha \to \beta) \to [\alpha] \to [\beta] \\
&\texttt{map}\ f\ [\,] \qquad\quad = [\,] \\
&\texttt{map}\ f\ (a : as) = (f\ a) : (\texttt{map}\ f\ as)
\end{aligned}$$

Using a Free Theorem [Wadler 1989]

For every

$$\texttt{get} :: [\alpha] \to [\alpha]$$

we have

$$\texttt{map}\ f\ (\texttt{get}\ l) \quad = \quad \texttt{get}\ (\texttt{map}\ f\ l)$$

for arbitrary f and l, where

$$\begin{array}{l}
\texttt{map} :: (\alpha \to \beta) \to [\alpha] \to [\beta] \\
\texttt{map}\ f\ [\,] \qquad\quad = [\,] \\
\texttt{map}\ f\ (a : as) = (f\ a) : (\texttt{map}\ f\ as)
\end{array}$$

But how do we know this?

Why `map` f (`get` l) $=$ `get` (`map` f l), Intuitively

- `get` $:: [\alpha] \to [\alpha]$ must work uniformly for every instantiation of α.

Why `map` f (`get` l) $=$ `get` (`map` f l), Intuitively

- `get` $:: [\alpha] \to [\alpha]$ must work uniformly for every instantiation of α.
- The output list can only contain elements from the input list l.

Why `map` f (`get` l) = `get` (`map` f l), Intuitively

- `get` $:: [\alpha] \to [\alpha]$ must work uniformly for every instantiation of α.
- The output list can only contain elements from the input list l.
- Which, and in which order/multiplicity, can only be decided based on l.

Why `map` f (`get` l) $=$ `get` (`map` f l), Intuitively

- `get` $:: [\alpha] \to [\alpha]$ must work uniformly for every instantiation of α.
- The output list can only contain elements from the input list l.
- Which, and in which order/multiplicity, can only be decided based on l.
- The only means for this decision is to inspect the length of l.

Why `map` f (`get` l) $=$ `get` (`map` f l), Intuitively

- `get` $:: [\alpha] \rightarrow [\alpha]$ must work uniformly for every instantiation of α.
- The output list can only contain elements from the input list l.
- Which, and in which order/multiplicity, can only be decided based on l.
- The only means for this decision is to inspect the length of l.
- The lists (`map` f l) and l always have equal length.

Why $\texttt{map}\ f\ (\texttt{get}\ l) = \texttt{get}\ (\texttt{map}\ f\ l)$, Intuitively

- $\texttt{get} :: [\alpha] \to [\alpha]$ must work uniformly for every instantiation of α.
- The output list can only contain elements from the input list l.
- Which, and in which order/multiplicity, can only be decided based on l.
- The only means for this decision is to inspect the length of l.
- The lists ($\texttt{map}\ f\ l$) and l always have equal length.
- $\texttt{get}$ always chooses "the same" elements from ($\texttt{map}\ f\ l$) for output as it does from l,

Why `map` f (`get` l) = `get` (`map` f l), Intuitively

- `get` :: $[\alpha] \to [\alpha]$ must work uniformly for every instantiation of α.
- The output list can only contain elements from the input list l.
- Which, and in which order/multiplicity, can only be decided based on l.
- The only means for this decision is to inspect the length of l.
- The lists (`map` f l) and l always have equal length.
- `get` always chooses "the same" elements from (`map` f l) for output as it does from l, except that in the former case it outputs their images under f.

Why map f (get l) = get (map f l), Intuitively

- get :: $[\alpha] \to [\alpha]$ must work uniformly for every instantiation of α.
- The output list can only contain elements from the input list l.
- Which, and in which order/multiplicity, can only be decided based on l.
- The only means for this decision is to inspect the length of l.
- The lists (map f l) and l always have equal length.
- get always chooses "the same" elements from (map f l) for output as it does from l, except that in the former case it outputs their images under f.
- (get (map f l)) is equivalent to (map f (get l)).

Why `map` f (`get` l) = `get` (`map` f l), Intuitively

- `get` :: $[\alpha] \to [\alpha]$ must work uniformly for every instantiation of α.
- The output list can only contain elements from the input list l.
- Which, and in which order/multiplicity, can only be decided based on l.
- The only means for this decision is to inspect the length of l.
- The lists (`map` f l) and l always have equal length.
- `get` always chooses "the same" elements from (`map` f l) for output as it does from l, except that in the former case it outputs their images under f.
- (`get` (`map` f l)) is equivalent to (`map` f (`get` l)).
- That is what was claimed!

Another Example

$$
\begin{array}{l}
\texttt{takeWhile} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha] \\
\texttt{takeWhile}\ p\ [\,] \qquad\quad = [\,] \\
\texttt{takeWhile}\ p\ (a : as)\ \mid\ p\ a \qquad\qquad = a : (\texttt{takeWhile}\ p\ as) \\
\qquad\qquad\qquad\qquad\quad\ \mid\ \texttt{otherwise} = [\,]
\end{array}
$$

Another Example

$$
\begin{array}{l}
\texttt{takeWhile} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha] \\
\texttt{takeWhile}\ p\ [\,] \qquad\quad = [\,] \\
\texttt{takeWhile}\ p\ (a : as)\ \mid\ p\ a \qquad\quad = a : (\texttt{takeWhile}\ p\ as) \\
\qquad\qquad\qquad\qquad\quad\ \mid\ \texttt{otherwise} = [\,]
\end{array}
$$

For arbitrary p, f, and l:

$$\texttt{takeWhile}\ p\ (\texttt{map}\ f\ \ l) \ \ = \ \ \texttt{map}\ f\ (\texttt{takeWhile}\ (p \circ f)\ l)$$

Provable by induction.

Another Example

$$
\begin{array}{l}
\texttt{takeWhile} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha] \\
\texttt{takeWhile}\ p\ [\,] \qquad\quad = [\,] \\
\texttt{takeWhile}\ p\ (a : as) \mid p\ a \qquad\quad\ \ = a : (\texttt{takeWhile}\ p\ as) \\
\qquad\qquad\qquad\qquad\quad\ \mid \texttt{otherwise} = [\,]
\end{array}
$$

For arbitrary p, f, and l:

$$\texttt{takeWhile}\ p\ (\texttt{map}\ f\ \ l) \;\;=\;\; \texttt{map}\ f\ (\texttt{takeWhile}\ (p \circ f)\ l)$$

Provable by induction.

Or again as a free theorem.

Another Example

$$\texttt{takeWhile} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$$

For arbitrary p, f, and l:

$$\texttt{takeWhile}\ p\ (\texttt{map}\ f\ l) \;=\; \texttt{map}\ f\ (\texttt{takeWhile}\ (p \circ f)\ l)$$

Provable by induction.

Or again as a free theorem.

Another Example

$$\texttt{takeWhile} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$$

$$\texttt{filter} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$$

For arbitrary p, f, and l:

$$\texttt{takeWhile}\ p\ (\texttt{map}\ f\ \ l) \quad = \quad \texttt{map}\ f\ (\texttt{takeWhile}\ (p \circ f)\ \ l)$$

$$\texttt{filter}\ p\ (\texttt{map}\ f\ \ l) \quad = \quad \texttt{map}\ f\ (\texttt{filter}\ (p \circ f)\ \ l)$$

Another Example

$$\texttt{takeWhile} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$$

$$\texttt{filter} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$$

$$\texttt{g} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$$

For arbitrary p, f, and l:

$$\texttt{takeWhile}\ p\ (\texttt{map}\ f\ \ l) \quad = \quad \texttt{map}\ f\ (\texttt{takeWhile}\ (p \circ f)\ \ l)$$

$$\texttt{filter}\ p\ (\texttt{map}\ f\ \ l) \quad = \quad \texttt{map}\ f\ (\texttt{filter}\ (p \circ f)\ \ l)$$

$$\texttt{g}\ p\ (\texttt{map}\ f\ \ l) \quad = \quad \texttt{map}\ f\ (\texttt{g}\ (p \circ f)\ \ l)$$

Why $\texttt{g}\ p\ (\texttt{map}\ f\ l) = \texttt{map}\ f\ (\texttt{g}\ (p \circ f)\ l)$, Intuitively

- $\texttt{g} :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$ must work uniformly for every instantiation of α.

Why $\texttt{g}\ p\ (\texttt{map}\ f\ l) = \texttt{map}\ f\ (\texttt{g}\ (p \circ f)\ l)$, Intuitively

- $\texttt{g} :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$ must work uniformly for every instantiation of α.
- The output list can only contain elements from the input list l.

Why $\texttt{g}\ p\ (\texttt{map}\ f\ l) = \texttt{map}\ f\ (\texttt{g}\ (p \circ f)\ l)$, Intuitively

- $\texttt{g} :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$ must work uniformly for every instantiation of α.
- The output list can only contain elements from the input list l.
- Which, and in which order/multiplicity, can only be decided based on l and the input predicate p.

Why $\texttt{g}\ p\ (\texttt{map}\ f\ l) = \texttt{map}\ f\ (\texttt{g}\ (p \circ f)\ l)$, Intuitively

- $\texttt{g} :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$ must work uniformly for every instantiation of α.
- The output list can only contain elements from the input list l.
- Which, and in which order/multiplicity, can only be decided based on l and the input predicate p.
- The only means for this decision are to inspect the length of l and to check the outcome of p on its elements.

Why $\texttt{g}\ p\ (\texttt{map}\ f\ l) = \texttt{map}\ f\ (\texttt{g}\ (p \circ f)\ l)$, Intuitively

- $\texttt{g} :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$ must work uniformly for every instantiation of α.
- The output list can only contain elements from the input list l.
- Which, and in which order/multiplicity, can only be decided based on l and the input predicate p.
- The only means for this decision are to inspect the length of l and to check the outcome of p on its elements.
- The lists $(\texttt{map}\ f\ l)$ and l always have equal length.

Why g p (map f l) $=$ map f (g ($p \circ f$) l), Intuitively

- g :: $(\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$ must work uniformly for every instantiation of α.
- The output list can only contain elements from the input list l.
- Which, and in which order/multiplicity, can only be decided based on l and the input predicate p.
- The only means for this decision are to inspect the length of l and to check the outcome of p on its elements.
- The lists (map f l) and l always have equal length.
- Applying p to an element of (map f l) always has the same outcome as applying ($p \circ f$) to the corresponding element of l.

Why $\mathtt{g}\ p\ (\mathtt{map}\ f\ l) = \mathtt{map}\ f\ (\mathtt{g}\ (p \circ f)\ l)$, Intuitively

- $\mathtt{g} :: (\alpha \to \mathrm{Bool}) \to [\alpha] \to [\alpha]$ must work uniformly for every instantiation of α.
- The output list can only contain elements from the input list l.
- Which, and in which order/multiplicity, can only be decided based on l and the input predicate p.
- The only means for this decision are to inspect the length of l and to check the outcome of p on its elements.
- The lists $(\mathtt{map}\ f\ l)$ and l always have equal length.
- Applying p to an element of $(\mathtt{map}\ f\ l)$ always has the same outcome as applying $(p \circ f)$ to the corresponding element of l.
- $\mathtt{g}$ with p always chooses "the same" elements from $(\mathtt{map}\ f\ l)$ for output as does $\mathtt{g}$ with $(p \circ f)$ from l, except that in the former case it outputs their images under f.

Why $\texttt{g}\ p\ (\texttt{map}\ f\ l) = \texttt{map}\ f\ (\texttt{g}\ (p \circ f)\ l)$, Intuitively

- $\texttt{g} :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$ must work uniformly for every instantiation of α.
- The output list can only contain elements from the input list l.
- Which, and in which order/multiplicity, can only be decided based on l and the input predicate p.
- The only means for this decision are to inspect the length of l and to check the outcome of p on its elements.
- The lists $(\texttt{map}\ f\ l)$ and l always have equal length.
- Applying p to an element of $(\texttt{map}\ f\ l)$ always has the same outcome as applying $(p \circ f)$ to the corresponding element of l.
- $\texttt{g}$ with p always chooses "the same" elements from $(\texttt{map}\ f\ l)$ for output as does $\texttt{g}$ with $(p \circ f)$ from l, except that in the former case it outputs their images under f.
- $(\texttt{g}\ p\ (\texttt{map}\ f\ l))$ is equivalent to $(\texttt{map}\ f\ (\texttt{g}\ (p \circ f)\ l))$.

Why $\mathtt{g}\ p\ (\mathtt{map}\ f\ l) = \mathtt{map}\ f\ (\mathtt{g}\ (p \circ f)\ l)$, Intuitively

- $\mathtt{g} :: (\alpha \to \mathrm{Bool}) \to [\alpha] \to [\alpha]$ must work uniformly for every instantiation of α.
- The output list can only contain elements from the input list l.
- Which, and in which order/multiplicity, can only be decided based on l and the input predicate p.
- The only means for this decision are to inspect the length of l and to check the outcome of p on its elements.
- The lists $(\mathtt{map}\ f\ l)$ and l always have equal length.
- Applying p to an element of $(\mathtt{map}\ f\ l)$ always has the same outcome as applying $(p \circ f)$ to the corresponding element of l.
- $\mathtt{g}$ with p always chooses "the same" elements from $(\mathtt{map}\ f\ l)$ for output as does $\mathtt{g}$ with $(p \circ f)$ from l, except that in the former case it outputs their images under f.
- $(\mathtt{g}\ p\ (\mathtt{map}\ f\ l))$ is equivalent to $(\mathtt{map}\ f\ (\mathtt{g}\ (p \circ f)\ l))$.
- That is what was claimed!

Automatic Generation of Free Theorems

At `http://www-ps.iai.uni-bonn.de/ft`:

This tool allows to generate free theorems for sublanguages of Haskell as described here.

The source code of the underlying library and a shell-based application using it is available here and here.

Please enter a (polymorphic) type, e.g. "(a -> Bool) -> [a] -> [a]" or simply "filter":

```
g :: (a -> Bool) -> [a] -> [a]
```

Please choose a sublanguage of Haskell:

- (•) no bottoms (hence no general recursion and no selective strictness)
- ( ) general recursion but no selective strictness
- ( ) general recursion and selective strictness

Please choose a theorem style (without effect in the sublanguage with no bottoms):

- (•) equational
- ( ) inequational

Generate

Automatic Generation of Free Theorems

The theorem generated for functions of the type

```
g :: forall a . (a -> Bool) -> [a] -> [a]
```

in the sublanguage of Haskell with no bottoms is:

```
forall t1,t2 in TYPES, R in REL(t1,t2).
 forall p :: t1 -> Bool.
  forall q :: t2 -> Bool.
   (forall (x, y) in R. p x = q y)
   ==> (forall (z, v) in lift{[]}(R).
         (g p z, g q v) in lift{[]}(R))
```

The structural lifting occurring therein is defined as follows:

```
lift{[]}(R)
  = {([], [])}
  u {(x : xs, y : ys) |
       ((x, y) in R) && ((xs, ys) in lift{[]}(R))}
```

Reducing all permissible relation variables to functions yields:

```
forall t1,t2 in TYPES, f :: t1 -> t2.
 forall p :: t1 -> Bool.
  forall q :: t2 -> Bool.
   (forall x :: t1. p x = q (f x))
   ==> (forall y :: [t1]. map f (g p y) = g q (map f y))
```

Export as PDF   Show type instantiations   Enter a new type   Help page

Formal Background: Parametric Polymorphism

Question: What g have type $\forall\alpha.\ (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$?

Formal Background: Parametric Polymorphism

Question: What g have type $\forall\alpha.\ (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$?

Approach: Give denotations of types as sets. (A bit naive . . .)

Formal Background: Parametric Polymorphism

Question: What g have type $\forall\alpha.\ (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$?

Approach: Give denotations of types as sets. (A bit naive . . .)

$$
\begin{array}{lcl}
[\![\mathsf{Bool}]\!] & = & \{\mathsf{True}, \mathsf{False}\} \\
[\![\mathsf{Int}]\!] & = & \{\ldots, -2, -1, 0, 1, 2, \ldots\}
\end{array}
$$

Formal Background: Parametric Polymorphism

Question: What g have type $\forall\alpha.\ (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$?

Approach: Give denotations of types as sets. (A bit naive ...)

$$
\begin{array}{lcl}
[\![\mathsf{Bool}]\!] & = & \{\mathsf{True}, \mathsf{False}\} \\
[\![\mathsf{Int}]\!] & = & \{\ldots, -2, -1, 0, 1, 2, \ldots\} \\
[\![(\tau_1, \tau_2)]\!] & = & [\![\tau_1]\!] \times [\![\tau_2]\!] \\
[\![[\tau]]\!] & = & \{[x_1, \ldots, x_n] \mid n \geq 0, x_i \in [\![\tau]\!]\}
\end{array}
$$

Formal Background: Parametric Polymorphism

Question: What g have type $\forall\alpha.\ (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$?

Approach: Give denotations of types as sets. (A bit naive . . .)

$$
\begin{array}{lcl}
[\![\mathsf{Bool}]\!] & = & \{\mathsf{True}, \mathsf{False}\} \\
[\![\mathsf{Int}]\!] & = & \{\dots, -2, -1, 0, 1, 2, \dots\} \\
[\![(\tau_1, \tau_2)]\!] & = & [\![\tau_1]\!] \times [\![\tau_2]\!] \\
[\![[\tau]]\!] & = & \{[x_1, \dots, x_n] \mid n \geq 0, x_i \in [\![\tau]\!]\} \\
[\![\tau_1 \to \tau_2]\!] & = & \{f : [\![\tau_1]\!] \to [\![\tau_2]\!]\}
\end{array}
$$

Formal Background: Parametric Polymorphism

Question: What g have type $\forall\alpha.\ (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$?

Approach: Give denotations of types as sets. (A bit naive . . .)

$$
\begin{array}{lcl}
[\![\mathsf{Bool}]\!] & = & \{\mathsf{True}, \mathsf{False}\} \\
[\![\mathsf{Int}]\!] & = & \{\ldots, -2, -1, 0, 1, 2, \ldots\} \\
[\![(\tau_1, \tau_2)]\!] & = & [\![\tau_1]\!] \times [\![\tau_2]\!] \\
[\![[\tau]]\!] & = & \{[x_1, \ldots, x_n] \mid n \geq 0, x_i \in [\![\tau]\!]\} \\
[\![\tau_1 \to \tau_2]\!] & = & \{f : [\![\tau_1]\!] \to [\![\tau_2]\!]\} \\
[\![\forall\alpha.\tau]\!] & = & ?
\end{array}
$$

Formal Background: Parametric Polymorphism

Question: What g have type $\forall\alpha.\ (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$?

Approach: Give denotations of types as sets. (A bit naive ...)

$$
\begin{array}{lcl}
[\![\mathsf{Bool}]\!] & = & \{\mathsf{True}, \mathsf{False}\} \\
[\![\mathsf{Int}]\!] & = & \{\ldots, -2, -1, 0, 1, 2, \ldots\} \\
[\![(\tau_1, \tau_2)]\!] & = & [\![\tau_1]\!] \times [\![\tau_2]\!] \\
[\![[\tau]]\!] & = & \{[x_1, \ldots, x_n] \mid n \geq 0, x_i \in [\![\tau]\!]\} \\
[\![\tau_1 \to \tau_2]\!] & = & \{f : [\![\tau_1]\!] \to [\![\tau_2]\!]\} \\
[\![\forall\alpha.\tau]\!] & = & ?
\end{array}
$$

- $g \in [\![\forall\alpha.\tau]\!]$ would have to be a whole "collection" of values: for every type τ', an instance with type $\tau[\tau'/\alpha]$.

Formal Background: Parametric Polymorphism

Question: What g have type $\forall\alpha.\ (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$?

Approach: Give denotations of types as sets. (A bit naive ...)

$$
\begin{array}{lcl}
[\![\mathsf{Bool}]\!] & = & \{\mathsf{True}, \mathsf{False}\} \\
[\![\mathsf{Int}]\!] & = & \{\dots, -2, -1, 0, 1, 2, \dots\} \\
[\![(\tau_1, \tau_2)]\!] & = & [\![\tau_1]\!] \times [\![\tau_2]\!] \\
[\![[\tau]]\!] & = & \{[x_1, \dots, x_n] \mid n \geq 0, x_i \in [\![\tau]\!]\} \\
[\![\tau_1 \to \tau_2]\!] & = & \{f : [\![\tau_1]\!] \to [\![\tau_2]\!]\} \\
[\![\forall\alpha.\tau]\!] & = & ?
\end{array}
$$

- $g \in [\![\forall\alpha.\tau]\!]$ would have to be a whole "collection" of values: for every type τ', an instance with type $\tau[\tau'/\alpha]$.
- $[\![\forall\alpha.\tau]\!] = \{g \mid \forall\tau'.\ g_{\tau'} \in [\![\tau[\tau'/\alpha]]\!]\}$?

Formal Background: Parametric Polymorphism

Question: What g have type $\forall\alpha.\ (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$?

Approach: Give denotations of types as sets. (A bit naive . . .)

$$
\begin{array}{lcl}
[\![\mathsf{Bool}]\!] & = & \{\mathsf{True}, \mathsf{False}\} \\
[\![\mathsf{Int}]\!] & = & \{\ldots, -2, -1, 0, 1, 2, \ldots\} \\
[\![(\tau_1, \tau_2)]\!] & = & [\![\tau_1]\!] \times [\![\tau_2]\!] \\
[\![[\tau]]\!] & = & \{[x_1, \ldots, x_n] \mid n \geq 0, x_i \in [\![\tau]\!]\} \\
[\![\tau_1 \to \tau_2]\!] & = & \{f : [\![\tau_1]\!] \to [\![\tau_2]\!]\} \\
[\![\forall\alpha.\tau]\!] & = & ?
\end{array}
$$

- $g \in [\![\forall\alpha.\tau]\!]$ would have to be a whole "collection" of values: for every type τ', an instance with type $\tau[\tau'/\alpha]$.
- $[\![\forall\alpha.\tau]\!] = \{g \mid \forall\tau'.\ g_{\tau'} \in [\![\tau[\tau'/\alpha]]\!]\}$?
- But this includes "ad-hoc polymorphic" functions!

Unwanted Ad-Hoc Polymorphism: Example

- With the proposed definition,
  $[\![\forall\alpha.\ (\alpha,\alpha) \to \alpha]\!] = \{g \mid \forall\tau.\ g_\tau : [\![\tau]\!] \times [\![\tau]\!] \to [\![\tau]\!]\}$.

Unwanted Ad-Hoc Polymorphism: Example

- With the proposed definition, $[\![\forall\alpha.\ (\alpha,\alpha) \to \alpha]\!] = \{g \mid \forall\tau.\ g_\tau : [\![\tau]\!] \times [\![\tau]\!] \to [\![\tau]\!]\}$.
- But this also allows a g with

$$\begin{array}{lcl} g_{\mathsf{Bool}}\ (x,y) & = & \texttt{not}\ x \\ g_{\mathsf{Int}}\ \ (x,y) & = & y+1\,, \end{array}$$

which is not possible in Haskell at type $\forall\alpha.\ (\alpha,\alpha) \to \alpha$.

Unwanted Ad-Hoc Polymorphism: Example

- With the proposed definition, $[\![\forall\alpha.\ (\alpha,\alpha)\to\alpha]\!] = \{g \mid \forall\tau.\ g_\tau : [\![\tau]\!]\times[\![\tau]\!]\to[\![\tau]\!]\}$.
- But this also allows a g with

$$\begin{array}{lcl} g_{\mathsf{Bool}}\ (x,y) & = & \texttt{not}\ x \\ g_{\mathsf{Int}}\ \ (x,y) & = & y+1\,, \end{array}$$

  which is not possible in Haskell at type $\forall\alpha.\ (\alpha,\alpha)\to\alpha$.
- To prevent this, we have to compare

$$\begin{array}{lcl} g_{\mathsf{Bool}} & : & [\![\mathsf{Bool}]\!]\times[\![\mathsf{Bool}]\!]\to[\![\mathsf{Bool}]\!] \quad \text{and} \\ g_{\mathsf{Int}} & : & [\![\mathsf{Int}]\!]\times[\![\mathsf{Int}]\!]\to[\![\mathsf{Int}]\!]\,, \end{array}$$

  and ensure that they "behave identically".

Unwanted Ad-Hoc Polymorphism: Example

- With the proposed definition,
  $[\![\forall\alpha.\ (\alpha,\alpha)\to\alpha]\!] = \{g \mid \forall\tau.\ g_\tau : [\![\tau]\!]\times[\![\tau]\!]\to[\![\tau]\!]\}$.
- But this also allows a g with

$$\begin{aligned} g_{\mathsf{Bool}}\ (x,y) &= \texttt{not}\ x \\ g_{\mathsf{Int}}\ \ (x,y) &= y+1\,, \end{aligned}$$

  which is not possible in Haskell at type $\forall\alpha.\ (\alpha,\alpha)\to\alpha$.
- To prevent this, we have to compare

$$\begin{aligned} g_{\mathsf{Bool}} &: [\![\mathsf{Bool}]\!]\times[\![\mathsf{Bool}]\!]\to[\![\mathsf{Bool}]\!] \quad \text{and} \\ g_{\mathsf{Int}} &: [\![\mathsf{Int}]\!]\times[\![\mathsf{Int}]\!]\to[\![\mathsf{Int}]\!]\,, \end{aligned}$$

  and ensure that they “behave identically”.
  But how?

Key Idea [Reynolds 1983]

Use arbitrary relations to tie instances together!

Key Idea [Reynolds 1983]

Use arbitrary relations to tie instances together!

In the example ($g :: \forall\alpha.\ (\alpha,\alpha) \to \alpha$):

- Choose a relation $\mathcal{R} \subseteq [\![\mathsf{Bool}]\!] \times [\![\mathsf{Int}]\!]$.

Key Idea [Reynolds 1983]

Use arbitrary relations to tie instances together!

In the example ($g :: \forall\alpha.\ (\alpha, \alpha) \to \alpha$):

- Choose a relation $\mathcal{R} \subseteq [\![\mathsf{Bool}]\!] \times [\![\mathsf{Int}]\!]$.
- Call $(x_1, x_2) \in [\![\mathsf{Bool}]\!] \times [\![\mathsf{Bool}]\!]$ and $(y_1, y_2) \in [\![\mathsf{Int}]\!] \times [\![\mathsf{Int}]\!]$ related if $(x_1, y_1) \in \mathcal{R}$ and $(x_2, y_2) \in \mathcal{R}$.

Key Idea [Reynolds 1983]

Use arbitrary relations to tie instances together!

In the example ($g :: \forall\alpha.\ (\alpha, \alpha) \to \alpha$):

- Choose a relation $\mathcal{R} \subseteq [\![\mathsf{Bool}]\!] \times [\![\mathsf{Int}]\!]$.
- Call $(x_1, x_2) \in [\![\mathsf{Bool}]\!] \times [\![\mathsf{Bool}]\!]$ and $(y_1, y_2) \in [\![\mathsf{Int}]\!] \times [\![\mathsf{Int}]\!]$ related if $(x_1, y_1) \in \mathcal{R}$ and $(x_2, y_2) \in \mathcal{R}$.
- Call $f_1 : [\![\mathsf{Bool}]\!] \times [\![\mathsf{Bool}]\!] \to [\![\mathsf{Bool}]\!]$, $f_2 : [\![\mathsf{Int}]\!] \times [\![\mathsf{Int}]\!] \to [\![\mathsf{Int}]\!]$ related if related inputs lead to related outputs.

Key Idea [Reynolds 1983]

Use arbitrary relations to tie instances together!

In the example ($\mathtt{g} :: \forall\alpha.\ (\alpha, \alpha) \to \alpha$):

- Choose a relation $\mathcal{R} \subseteq [\![\mathsf{Bool}]\!] \times [\![\mathsf{Int}]\!]$.
- Call $(x_1, x_2) \in [\![\mathsf{Bool}]\!] \times [\![\mathsf{Bool}]\!]$ and $(y_1, y_2) \in [\![\mathsf{Int}]\!] \times [\![\mathsf{Int}]\!]$ related if $(x_1, y_1) \in \mathcal{R}$ and $(x_2, y_2) \in \mathcal{R}$.
- Call $f_1 : [\![\mathsf{Bool}]\!] \times [\![\mathsf{Bool}]\!] \to [\![\mathsf{Bool}]\!]$, $f_2 : [\![\mathsf{Int}]\!] \times [\![\mathsf{Int}]\!] \to [\![\mathsf{Int}]\!]$ related if related inputs lead to related outputs.
- Then g_{Bool} and g_{Int} with

$$\begin{aligned} g_{\mathsf{Bool}}\ (x, y) &= \mathtt{not}\ x \\ g_{\mathsf{Int}}\ (x, y) &= y + 1 \end{aligned}$$

  are not related for choice of, e.g., $\mathcal{R} = \{(\mathsf{True}, 1)\}$.

Key Idea [Reynolds 1983]

Use arbitrary relations to tie instances together!

In the example ($\mathtt{g} :: \forall\alpha.\ (\alpha,\alpha) \to \alpha$):

- Choose a relation $\mathcal{R} \subseteq [\![\mathsf{Bool}]\!] \times [\![\mathsf{Int}]\!]$.
- Call $(x_1, x_2) \in [\![\mathsf{Bool}]\!] \times [\![\mathsf{Bool}]\!]$ and $(y_1, y_2) \in [\![\mathsf{Int}]\!] \times [\![\mathsf{Int}]\!]$ related if $(x_1, y_1) \in \mathcal{R}$ and $(x_2, y_2) \in \mathcal{R}$.
- Call $f_1 : [\![\mathsf{Bool}]\!] \times [\![\mathsf{Bool}]\!] \to [\![\mathsf{Bool}]\!]$, $f_2 : [\![\mathsf{Int}]\!] \times [\![\mathsf{Int}]\!] \to [\![\mathsf{Int}]\!]$ related if related inputs lead to related outputs.
- Then g_{Bool} and g_{Int} with

$$\begin{aligned} g_{\mathsf{Bool}}\ (x, y) &= \mathtt{not}\ x \\ g_{\mathsf{Int}}\ (x, y) &= y + 1 \end{aligned}$$

  are not related for choice of, e.g., $\mathcal{R} = \{(\mathsf{True}, 1)\}$.

Reynolds: $g \in [\![\forall\alpha.\tau]\!]$ iff for every τ_1, τ_2 and $\mathcal{R} \subseteq [\![\tau_1]\!] \times [\![\tau_2]\!]$,
g_{τ_1} is related to g_{τ_2} by the "propagation" of $\mathcal{R}$
along τ.

Polymorphic Lambda Calculus [Girard 1972, Reynolds 1974]

Types: $\tau := \alpha \mid \tau \to \tau \mid \forall\alpha.\tau$

Terms: $t := x \mid \lambda x : \tau.t \mid t\ t \mid \Lambda\alpha.t \mid t\ \tau$

Polymorphic Lambda Calculus [Girard 1972, Reynolds 1974]

Types: $\tau := \alpha \mid \tau \to \tau \mid \forall\alpha.\tau$

Terms: $t := x \mid \lambda x : \tau.t \mid t\ t \mid \Lambda\alpha.t \mid t\ \tau$

$$\Gamma, x : \tau \vdash x : \tau$$

Polymorphic Lambda Calculus [Girard 1972, Reynolds 1974]

Types: $\tau := \alpha \mid \tau \to \tau \mid \forall \alpha . \tau$

Terms: $t := x \mid \lambda x : \tau . t \mid t \; t \mid \Lambda \alpha . t \mid t \; \tau$

$$\Gamma, x : \tau \vdash x : \tau$$

$$\frac{\Gamma, x : \tau_1 \vdash t : \tau_2}{\Gamma \vdash (\lambda x : \tau_1 . t) : \tau_1 \to \tau_2}$$

Polymorphic Lambda Calculus [Girard 1972, Reynolds 1974]

Types: $\tau := \alpha \mid \tau \to \tau \mid \forall \alpha . \tau$

Terms: $t := x \mid \lambda x : \tau . t \mid t\ t \mid \Lambda \alpha . t \mid t\ \tau$

$$\Gamma, x : \tau \vdash x : \tau$$

$$\frac{\Gamma, x : \tau_1 \vdash t : \tau_2}{\Gamma \vdash (\lambda x : \tau_1 . t) : \tau_1 \to \tau_2}$$

$$\frac{\Gamma \vdash t : \tau_1 \to \tau_2 \qquad \Gamma \vdash u : \tau_1}{\Gamma \vdash (t\ u) : \tau_2}$$

Polymorphic Lambda Calculus [Girard 1972, Reynolds 1974]

Types: $\tau := \alpha \mid \tau \to \tau \mid \forall\alpha.\tau$

Terms: $t := x \mid \lambda x : \tau.t \mid t\ t \mid \Lambda\alpha.t \mid t\ \tau$

$$\Gamma, x : \tau \vdash x : \tau$$

$$\frac{\Gamma, x : \tau_1 \vdash t : \tau_2}{\Gamma \vdash (\lambda x : \tau_1.t) : \tau_1 \to \tau_2}$$

$$\frac{\Gamma \vdash t : \tau_1 \to \tau_2 \qquad \Gamma \vdash u : \tau_1}{\Gamma \vdash (t\ u) : \tau_2}$$

$$\frac{\alpha, \Gamma \vdash t : \tau}{\Gamma \vdash (\Lambda\alpha.t) : \forall\alpha.\tau}$$

Polymorphic Lambda Calculus [Girard 1972, Reynolds 1974]

Types: $\tau := \alpha \mid \tau \to \tau \mid \forall \alpha . \tau$

Terms: $t := x \mid \lambda x : \tau . t \mid t \ t \mid \Lambda \alpha . t \mid t \ \tau$

$$\Gamma, x : \tau \vdash x : \tau$$

$$\frac{\Gamma, x : \tau_1 \vdash t : \tau_2}{\Gamma \vdash (\lambda x : \tau_1 . t) : \tau_1 \to \tau_2}$$

$$\frac{\Gamma \vdash t : \tau_1 \to \tau_2 \qquad \Gamma \vdash u : \tau_1}{\Gamma \vdash (t \ u) : \tau_2}$$

$$\frac{\alpha, \Gamma \vdash t : \tau}{\Gamma \vdash (\Lambda \alpha . t) : \forall \alpha . \tau}$$

$$\frac{\Gamma \vdash t : \forall \alpha . \tau}{\Gamma \vdash (t \ \tau') : \tau[\tau' / \alpha]}$$

Polymorphic Lambda Calculus [Girard 1972, Reynolds 1974]

Types: $\tau := \alpha \mid \tau \to \tau \mid \forall \alpha.\tau$

Terms: $t := x \mid \lambda x : \tau.t \mid t\ t \mid \Lambda \alpha.t \mid t\ \tau$

$$\Gamma, x : \tau \vdash x : \tau \qquad [\![x]\!]_{\theta,\sigma} = \sigma(x)$$

$$\frac{\Gamma, x : \tau_1 \vdash t : \tau_2}{\Gamma \vdash (\lambda x : \tau_1.t) : \tau_1 \to \tau_2} \qquad [\![\lambda x : \tau_1.t]\!]_{\theta,\sigma}\ a = [\![t]\!]_{\theta,\sigma[x \mapsto a]}$$

$$\frac{\Gamma \vdash t : \tau_1 \to \tau_2 \qquad \Gamma \vdash u : \tau_1}{\Gamma \vdash (t\ u) : \tau_2} \qquad [\![t\ u]\!]_{\theta,\sigma} = [\![t]\!]_{\theta,\sigma}\ [\![u]\!]_{\theta,\sigma}$$

$$\frac{\alpha, \Gamma \vdash t : \tau}{\Gamma \vdash (\Lambda \alpha.t) : \forall \alpha.\tau} \qquad [\![\Lambda \alpha.t]\!]_{\theta,\sigma}\ S = [\![t]\!]_{\theta[\alpha \mapsto S],\sigma}$$

$$\frac{\Gamma \vdash t : \forall \alpha.\tau}{\Gamma \vdash (t\ \tau') : \tau[\tau'/\alpha]} \qquad [\![t\ \tau']\!]_{\theta,\sigma} = [\![t]\!]_{\theta,\sigma}\ [\![\tau']\!]_{\theta}$$

Parametricity Theorem [Reynolds 1983, Wadler 1989]

Given τ and environments θ_1, θ_2, ρ with $\rho(\alpha) \subseteq \theta_1(\alpha) \times \theta_2(\alpha)$, define $\Delta_{\tau,\rho} \subseteq [\![\tau]\!]_{\theta_1} \times [\![\tau]\!]_{\theta_2}$ as follows:

Parametricity Theorem [Reynolds 1983, Wadler 1989]

Given τ and environments θ_1, θ_2, ρ with $\rho(\alpha) \subseteq \theta_1(\alpha) \times \theta_2(\alpha)$, define $\Delta_{\tau,\rho} \subseteq [\![\tau]\!]_{\theta_1} \times [\![\tau]\!]_{\theta_2}$ as follows:

$$\Delta_{\alpha,\rho} \quad = \quad \rho(\alpha)$$

Parametricity Theorem [Reynolds 1983, Wadler 1989]

Given τ and environments θ_1, θ_2, ρ with $\rho(\alpha) \subseteq \theta_1(\alpha) \times \theta_2(\alpha)$, define $\Delta_{\tau,\rho} \subseteq [\![\tau]\!]_{\theta_1} \times [\![\tau]\!]_{\theta_2}$ as follows:

$$
\begin{array}{lcl}
\Delta_{\alpha,\rho} & = & \rho(\alpha) \\
\Delta_{\tau_1 \to \tau_2,\rho} & = & \{(f_1, f_2) \mid \forall (a_1, a_2) \in \Delta_{\tau_1,\rho}.\ (f_1\ a_1, f_2\ a_2) \in \Delta_{\tau_2,\rho}\}
\end{array}
$$

Parametricity Theorem [Reynolds 1983, Wadler 1989]

Given τ and environments θ_1, θ_2, ρ with $\rho(\alpha) \subseteq \theta_1(\alpha) \times \theta_2(\alpha)$, define $\Delta_{\tau,\rho} \subseteq [\![\tau]\!]_{\theta_1} \times [\![\tau]\!]_{\theta_2}$ as follows:

$$
\begin{array}{lcl}
\Delta_{\alpha,\rho} & = & \rho(\alpha) \\
\Delta_{\tau_1 \to \tau_2,\rho} & = & \{(f_1, f_2) \mid \forall (a_1, a_2) \in \Delta_{\tau_1,\rho}.\ (f_1\ a_1, f_2\ a_2) \in \Delta_{\tau_2,\rho}\} \\
\Delta_{\forall \alpha.\tau,\rho} & = & \{(g_1, g_2) \mid \forall \mathcal{R} \subseteq S_1 \times S_2.\ (g_1\ S_1, g_2\ S_2) \in \Delta_{\tau,\rho[\alpha \mapsto \mathcal{R}]}\}
\end{array}
$$

Parametricity Theorem [Reynolds 1983, Wadler 1989]

Given τ and environments θ_1, θ_2, ρ with $\rho(\alpha) \subseteq \theta_1(\alpha) \times \theta_2(\alpha)$, define $\Delta_{\tau,\rho} \subseteq [\![\tau]\!]_{\theta_1} \times [\![\tau]\!]_{\theta_2}$ as follows:

$$\begin{array}{lcl}
\Delta_{\alpha,\rho} & = & \rho(\alpha) \\
\Delta_{\tau_1 \to \tau_2,\rho} & = & \{(f_1, f_2) \mid \forall (a_1, a_2) \in \Delta_{\tau_1,\rho}.\ (f_1\ a_1, f_2\ a_2) \in \Delta_{\tau_2,\rho}\} \\
\Delta_{\forall \alpha.\tau,\rho} & = & \{(g_1, g_2) \mid \forall \mathcal{R} \subseteq S_1 \times S_2.\ (g_1\ S_1, g_2\ S_2) \in \Delta_{\tau,\rho[\alpha \mapsto \mathcal{R}]}\}
\end{array}$$

Then, for every closed term t of closed type τ:

$$([\![t]\!]_{\emptyset,\emptyset}, [\![t]\!]_{\emptyset,\emptyset}) \in \Delta_{\tau,\emptyset}.$$

Proof Sketch

Prove the following more general statement:

> $\Gamma \vdash t : \tau$ implies $(\llbracket t \rrbracket_{\theta_1,\sigma_1}, \llbracket t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau,\rho}$,
> provided $(\sigma_1(x), \sigma_2(x)) \in \Delta_{\tau',\rho}$ for every $x : \tau'$ in Γ

by induction on the structure of typing derivations.

Proof Sketch

Prove the following more general statement:

$\Gamma \vdash t : \tau$ implies $([\![t]\!]_{\theta_1,\sigma_1}, [\![t]\!]_{\theta_2,\sigma_2}) \in \Delta_{\tau,\rho}$,
provided $(\sigma_1(x), \sigma_2(x)) \in \Delta_{\tau',\rho}$ for every $x : \tau'$ in Γ

by induction on the structure of typing derivations.
The base case is immediate.

Proof Sketch

Prove the following more general statement:

$$\Gamma \vdash t : \tau \text{ implies } (\llbracket t \rrbracket_{\theta_1,\sigma_1}, \llbracket t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau,\rho} \ ,$$
$$\text{provided } (\sigma_1(x), \sigma_2(x)) \in \Delta_{\tau',\rho} \text{ for every } x : \tau' \text{ in } \Gamma$$

by induction on the structure of typing derivations.

The base case is immediate. In the step cases:

$$\frac{\Gamma, x : \tau_1 \vdash t : \tau_2}{\Gamma \vdash (\lambda x : \tau_1 . t) : \tau_1 \to \tau_2}$$

Proof Sketch

Prove the following more general statement:

$\Gamma \vdash t : \tau$ implies $(\llbracket t \rrbracket_{\theta_1,\sigma_1}, \llbracket t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau,\rho}$,
provided $(\sigma_1(x), \sigma_2(x)) \in \Delta_{\tau',\rho}$ for every $x : \tau'$ in Γ

by induction on the structure of typing derivations.
The base case is immediate. In the step cases:

$$\frac{\Gamma, x : \tau_1 \vdash t : \tau_2}{(\llbracket \lambda x : \tau_1 . t \rrbracket_{\theta_1,\sigma_1}, \llbracket \lambda x : \tau_1 . t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_1 \to \tau_2,\rho}}$$

Proof Sketch

Prove the following more general statement:

$\Gamma \vdash t : \tau$ implies $([\![t]\!]_{\theta_1,\sigma_1}, [\![t]\!]_{\theta_2,\sigma_2}) \in \Delta_{\tau,\rho}$,
provided $(\sigma_1(x), \sigma_2(x)) \in \Delta_{\tau',\rho}$ for every $x : \tau'$ in Γ

by induction on the structure of typing derivations.
The base case is immediate. In the step cases:

$$\frac{\forall (a_1, a_2) \in \Delta_{\tau_1,\rho}.\ ([\![t]\!]_{\theta_1,\sigma_1[x\mapsto a_1]}, [\![t]\!]_{\theta_2,\sigma_2[x\mapsto a_2]}) \in \Delta_{\tau_2,\rho}}{([\![\lambda x : \tau_1.t]\!]_{\theta_1,\sigma_1}, [\![\lambda x : \tau_1.t]\!]_{\theta_2,\sigma_2}) \in \Delta_{\tau_1\to\tau_2,\rho}}$$

Proof Sketch

Prove the following more general statement:

$\Gamma \vdash t : \tau$ implies $(\llbracket t \rrbracket_{\theta_1,\sigma_1}, \llbracket t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau,\rho}$,
provided $(\sigma_1(x), \sigma_2(x)) \in \Delta_{\tau',\rho}$ for every $x : \tau'$ in Γ

by induction on the structure of typing derivations.
The base case is immediate. In the step cases:

$$\frac{\forall (a_1, a_2) \in \Delta_{\tau_1,\rho}.\ (\llbracket t \rrbracket_{\theta_1,\sigma_1[x\mapsto a_1]}, \llbracket t \rrbracket_{\theta_2,\sigma_2[x\mapsto a_2]}) \in \Delta_{\tau_2,\rho}}{(\llbracket \lambda x : \tau_1 . t \rrbracket_{\theta_1,\sigma_1}, \llbracket \lambda x : \tau_1 . t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_1 \to \tau_2,\rho}}$$

$$\frac{\Gamma \vdash t : \tau_1 \to \tau_2 \qquad \Gamma \vdash u : \tau_1}{\Gamma \vdash (t\ u) : \tau_2}$$

Proof Sketch

Prove the following more general statement:

$\Gamma \vdash t : \tau$ implies $(\llbracket t \rrbracket_{\theta_1,\sigma_1}, \llbracket t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau,\rho}$,
provided $(\sigma_1(x), \sigma_2(x)) \in \Delta_{\tau',\rho}$ for every $x : \tau'$ in Γ

by induction on the structure of typing derivations.
The base case is immediate. In the step cases:

$$\frac{\forall (a_1, a_2) \in \Delta_{\tau_1,\rho}.\ (\llbracket t \rrbracket_{\theta_1,\sigma_1[x \mapsto a_1]}, \llbracket t \rrbracket_{\theta_2,\sigma_2[x \mapsto a_2]}) \in \Delta_{\tau_2,\rho}}{(\llbracket \lambda x : \tau_1 . t \rrbracket_{\theta_1,\sigma_1}, \llbracket \lambda x : \tau_1 . t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_1 \to \tau_2,\rho}}$$

$$\frac{\Gamma \vdash t : \tau_1 \to \tau_2 \qquad \Gamma \vdash u : \tau_1}{(\llbracket t\ u \rrbracket_{\theta_1,\sigma_1}, \llbracket t\ u \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_2,\rho}}$$

Proof Sketch

Prove the following more general statement:

$\Gamma \vdash t : \tau$ implies $(\llbracket t \rrbracket_{\theta_1,\sigma_1}, \llbracket t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau,\rho}$,
provided $(\sigma_1(x), \sigma_2(x)) \in \Delta_{\tau',\rho}$ for every $x : \tau'$ in Γ

by induction on the structure of typing derivations.
The base case is immediate. In the step cases:

$$\frac{\forall (a_1, a_2) \in \Delta_{\tau_1,\rho}.\ (\llbracket t \rrbracket_{\theta_1,\sigma_1[x \mapsto a_1]}, \llbracket t \rrbracket_{\theta_2,\sigma_2[x \mapsto a_2]}) \in \Delta_{\tau_2,\rho}}{(\llbracket \lambda x : \tau_1 . t \rrbracket_{\theta_1,\sigma_1}, \llbracket \lambda x : \tau_1 . t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_1 \to \tau_2,\rho}}$$

$$\frac{(\llbracket t \rrbracket_{\theta_1,\sigma_1}, \llbracket t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_1 \to \tau_2,\rho} \qquad (\llbracket u \rrbracket_{\theta_1,\sigma_1}, \llbracket u \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_1,\rho}}{(\llbracket t\ u \rrbracket_{\theta_1,\sigma_1}, \llbracket t\ u \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_2,\rho}}$$

Proof Sketch

Prove the following more general statement:

$\Gamma \vdash t : \tau$ implies $([\![t]\!]_{\theta_1,\sigma_1}, [\![t]\!]_{\theta_2,\sigma_2}) \in \Delta_{\tau,\rho}$,
provided $(\sigma_1(x), \sigma_2(x)) \in \Delta_{\tau',\rho}$ for every $x : \tau'$ in Γ

by induction on the structure of typing derivations.
The base case is immediate. In the step cases:

$$\frac{\forall (a_1, a_2) \in \Delta_{\tau_1,\rho}.\ ([\![t]\!]_{\theta_1,\sigma_1[x \mapsto a_1]}, [\![t]\!]_{\theta_2,\sigma_2[x \mapsto a_2]}) \in \Delta_{\tau_2,\rho}}{([\![\lambda x : \tau_1.t]\!]_{\theta_1,\sigma_1}, [\![\lambda x : \tau_1.t]\!]_{\theta_2,\sigma_2}) \in \Delta_{\tau_1 \to \tau_2,\rho}}$$

$$\frac{([\![t]\!]_{\theta_1,\sigma_1}, [\![t]\!]_{\theta_2,\sigma_2}) \in \Delta_{\tau_1 \to \tau_2,\rho} \qquad ([\![u]\!]_{\theta_1,\sigma_1}, [\![u]\!]_{\theta_2,\sigma_2}) \in \Delta_{\tau_1,\rho}}{([\![t\ u]\!]_{\theta_1,\sigma_1}, [\![t\ u]\!]_{\theta_2,\sigma_2}) \in \Delta_{\tau_2,\rho}}$$

$$\frac{\alpha, \Gamma \vdash t : \tau}{\Gamma \vdash (\Lambda\alpha.t) : \forall\alpha.\tau}$$

Proof Sketch

Prove the following more general statement:

$\Gamma \vdash t : \tau$ implies $(\llbracket t \rrbracket_{\theta_1,\sigma_1}, \llbracket t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau,\rho}$,
provided $(\sigma_1(x), \sigma_2(x)) \in \Delta_{\tau',\rho}$ for every $x : \tau'$ in Γ

by induction on the structure of typing derivations.
The base case is immediate. In the step cases:

$$\frac{\forall (a_1, a_2) \in \Delta_{\tau_1,\rho}.\ (\llbracket t \rrbracket_{\theta_1,\sigma_1[x \mapsto a_1]}, \llbracket t \rrbracket_{\theta_2,\sigma_2[x \mapsto a_2]}) \in \Delta_{\tau_2,\rho}}{(\llbracket \lambda x : \tau_1.t \rrbracket_{\theta_1,\sigma_1}, \llbracket \lambda x : \tau_1.t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_1 \to \tau_2,\rho}}$$

$$\frac{(\llbracket t \rrbracket_{\theta_1,\sigma_1}, \llbracket t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_1 \to \tau_2,\rho} \qquad (\llbracket u \rrbracket_{\theta_1,\sigma_1}, \llbracket u \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_1,\rho}}{(\llbracket t\ u \rrbracket_{\theta_1,\sigma_1}, \llbracket t\ u \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_2,\rho}}$$

$$\frac{\alpha, \Gamma \vdash t : \tau}{(\llbracket \Lambda\alpha.t \rrbracket_{\theta_1,\sigma_1}, \llbracket \Lambda\alpha.t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\forall\alpha.\tau,\rho}}$$

Proof Sketch

Prove the following more general statement:

$\Gamma \vdash t : \tau$ implies $(\llbracket t \rrbracket_{\theta_1,\sigma_1}, \llbracket t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau,\rho}$,
provided $(\sigma_1(x), \sigma_2(x)) \in \Delta_{\tau',\rho}$ for every $x : \tau'$ in Γ

by induction on the structure of typing derivations.
The base case is immediate. In the step cases:

$$\frac{\forall (a_1, a_2) \in \Delta_{\tau_1,\rho}.\ (\llbracket t \rrbracket_{\theta_1,\sigma_1[x \mapsto a_1]}, \llbracket t \rrbracket_{\theta_2,\sigma_2[x \mapsto a_2]}) \in \Delta_{\tau_2,\rho}}{(\llbracket \lambda x : \tau_1 . t \rrbracket_{\theta_1,\sigma_1}, \llbracket \lambda x : \tau_1 . t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_1 \to \tau_2,\rho}}$$

$$\frac{(\llbracket t \rrbracket_{\theta_1,\sigma_1}, \llbracket t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_1 \to \tau_2,\rho} \qquad (\llbracket u \rrbracket_{\theta_1,\sigma_1}, \llbracket u \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_1,\rho}}{(\llbracket t\ u \rrbracket_{\theta_1,\sigma_1}, \llbracket t\ u \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_2,\rho}}$$

$$\frac{\forall \mathcal{R} \subseteq S_1 \times S_2.\ (\llbracket t \rrbracket_{\theta_1[\alpha \mapsto S_1],\sigma_1}, \llbracket t \rrbracket_{\theta_2[\alpha \mapsto S_2],\sigma_2}) \in \Delta_{\tau,\rho[\alpha \mapsto \mathcal{R}]}}{(\llbracket \Lambda \alpha . t \rrbracket_{\theta_1,\sigma_1}, \llbracket \Lambda \alpha . t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\forall \alpha . \tau,\rho}}$$

Proof Sketch

Prove the following more general statement:

$\Gamma \vdash t : \tau$ implies $(\llbracket t \rrbracket_{\theta_1,\sigma_1}, \llbracket t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau,\rho}$,
provided $(\sigma_1(x), \sigma_2(x)) \in \Delta_{\tau',\rho}$ for every $x : \tau'$ in Γ

by induction on the structure of typing derivations.
The base case is immediate. In the step cases:

$$\frac{\forall (a_1, a_2) \in \Delta_{\tau_1,\rho}.\ (\llbracket t \rrbracket_{\theta_1,\sigma_1[x\mapsto a_1]}, \llbracket t \rrbracket_{\theta_2,\sigma_2[x\mapsto a_2]}) \in \Delta_{\tau_2,\rho}}{(\llbracket \lambda x : \tau_1 . t \rrbracket_{\theta_1,\sigma_1}, \llbracket \lambda x : \tau_1 . t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_1 \to \tau_2,\rho}}$$

$$\frac{(\llbracket t \rrbracket_{\theta_1,\sigma_1}, \llbracket t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_1 \to \tau_2,\rho} \qquad (\llbracket u \rrbracket_{\theta_1,\sigma_1}, \llbracket u \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_1,\rho}}{(\llbracket t\ u \rrbracket_{\theta_1,\sigma_1}, \llbracket t\ u \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_2,\rho}}$$

$$\frac{\forall \mathcal{R} \subseteq S_1 \times S_2.\ (\llbracket t \rrbracket_{\theta_1[\alpha\mapsto S_1],\sigma_1}, \llbracket t \rrbracket_{\theta_2[\alpha\mapsto S_2],\sigma_2}) \in \Delta_{\tau,\rho[\alpha\mapsto\mathcal{R}]}}{(\llbracket \Lambda\alpha . t \rrbracket_{\theta_1,\sigma_1}, \llbracket \Lambda\alpha . t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\forall\alpha.\tau,\rho}}$$

$$\frac{\Gamma \vdash t : \forall\alpha.\tau}{\Gamma \vdash (t\ \tau') : \tau[\tau'/\alpha]}$$

Proof Sketch

Prove the following more general statement:

$\Gamma \vdash t : \tau$ implies $(\llbracket t \rrbracket_{\theta_1,\sigma_1}, \llbracket t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau,\rho}$,
provided $(\sigma_1(x), \sigma_2(x)) \in \Delta_{\tau',\rho}$ for every $x : \tau'$ in Γ

by induction on the structure of typing derivations.
The base case is immediate. In the step cases:

$$\frac{\forall (a_1, a_2) \in \Delta_{\tau_1,\rho}.\ (\llbracket t \rrbracket_{\theta_1,\sigma_1[x\mapsto a_1]}, \llbracket t \rrbracket_{\theta_2,\sigma_2[x\mapsto a_2]}) \in \Delta_{\tau_2,\rho}}{(\llbracket \lambda x : \tau_1.t \rrbracket_{\theta_1,\sigma_1}, \llbracket \lambda x : \tau_1.t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_1\to\tau_2,\rho}}$$

$$\frac{(\llbracket t \rrbracket_{\theta_1,\sigma_1}, \llbracket t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_1\to\tau_2,\rho} \qquad (\llbracket u \rrbracket_{\theta_1,\sigma_1}, \llbracket u \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_1,\rho}}{(\llbracket t\ u \rrbracket_{\theta_1,\sigma_1}, \llbracket t\ u \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_2,\rho}}$$

$$\frac{\forall \mathcal{R} \subseteq S_1 \times S_2.\ (\llbracket t \rrbracket_{\theta_1[\alpha\mapsto S_1],\sigma_1}, \llbracket t \rrbracket_{\theta_2[\alpha\mapsto S_2],\sigma_2}) \in \Delta_{\tau,\rho[\alpha\mapsto\mathcal{R}]}}{(\llbracket \Lambda\alpha.t \rrbracket_{\theta_1,\sigma_1}, \llbracket \Lambda\alpha.t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\forall\alpha.\tau,\rho}}$$

$$\frac{\Gamma \vdash t : \forall\alpha.\tau}{(\llbracket t\ \tau' \rrbracket_{\theta_1,\sigma_1}, \llbracket t\ \tau' \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau[\tau'/\alpha],\rho}}$$

Proof Sketch

Prove the following more general statement:

$\Gamma \vdash t : \tau$ implies $(\llbracket t \rrbracket_{\theta_1,\sigma_1}, \llbracket t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau,\rho}$,
provided $(\sigma_1(x), \sigma_2(x)) \in \Delta_{\tau',\rho}$ for every $x : \tau'$ in Γ

by induction on the structure of typing derivations.
The base case is immediate. In the step cases:

$$\frac{\forall (a_1, a_2) \in \Delta_{\tau_1,\rho}.\ (\llbracket t \rrbracket_{\theta_1,\sigma_1[x\mapsto a_1]}, \llbracket t \rrbracket_{\theta_2,\sigma_2[x\mapsto a_2]}) \in \Delta_{\tau_2,\rho}}{(\llbracket \lambda x : \tau_1 . t \rrbracket_{\theta_1,\sigma_1}, \llbracket \lambda x : \tau_1 . t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_1 \to \tau_2,\rho}}$$

$$\frac{(\llbracket t \rrbracket_{\theta_1,\sigma_1}, \llbracket t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_1 \to \tau_2,\rho} \qquad (\llbracket u \rrbracket_{\theta_1,\sigma_1}, \llbracket u \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_1,\rho}}{(\llbracket t\ u \rrbracket_{\theta_1,\sigma_1}, \llbracket t\ u \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_2,\rho}}$$

$$\frac{\forall \mathcal{R} \subseteq S_1 \times S_2.\ (\llbracket t \rrbracket_{\theta_1[\alpha\mapsto S_1],\sigma_1}, \llbracket t \rrbracket_{\theta_2[\alpha\mapsto S_2],\sigma_2}) \in \Delta_{\tau,\rho[\alpha\mapsto\mathcal{R}]}}{(\llbracket \Lambda\alpha . t \rrbracket_{\theta_1,\sigma_1}, \llbracket \Lambda\alpha . t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\forall\alpha.\tau,\rho}}$$

$$\frac{(\llbracket t \rrbracket_{\theta_1,\sigma_1}, \llbracket t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\forall\alpha.\tau,\rho}}{(\llbracket t\ \tau' \rrbracket_{\theta_1,\sigma_1}, \llbracket t\ \tau' \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau[\tau'/\alpha],\rho}}$$

Adding Datatypes

Types: $\tau := \ \cdots \mid \mathsf{Bool} \mid [\tau]$

Terms: $t := \ \cdots \mid \mathsf{True} \mid \mathsf{False} \mid [\,]_\tau \mid t : t \mid \textbf{case } t \textbf{ of } \{\cdots\}$

Adding Datatypes

$$\text{Types: } \tau := \ \cdots \mid \mathsf{Bool} \mid [\tau]$$
$$\text{Terms: } t := \ \cdots \mid \mathsf{True} \mid \mathsf{False} \mid [\,]_\tau \mid t : t \mid \textbf{case } t \textbf{ of } \{\cdots\}$$

$$\Gamma \vdash \mathsf{True} : \mathsf{Bool} \quad , \quad \Gamma \vdash \mathsf{False} : \mathsf{Bool} \quad , \quad \Gamma \vdash [\,]_\tau : [\tau]$$

$$\frac{\Gamma \vdash t : \tau \qquad \Gamma \vdash u : [\tau]}{\Gamma \vdash (t : u) : [\tau]}$$

$$\frac{\Gamma \vdash t : \mathsf{Bool} \qquad \Gamma \vdash u : \tau \qquad \Gamma \vdash v : \tau}{\Gamma \vdash (\textbf{case } t \textbf{ of } \{\mathsf{True} \to u\,; \mathsf{False} \to v\}) : \tau}$$

$$\frac{\Gamma \vdash t : [\tau'] \qquad \Gamma \vdash u : \tau \qquad \Gamma, x_1 : \tau', x_2 : [\tau'] \vdash v : \tau}{\Gamma \vdash (\textbf{case } t \textbf{ of } \{[] \to u\,; (x_1 : x_2) \to v\}) : \tau}$$

Adding Datatypes

$$\begin{array}{rl} \text{Types:} & \tau := \ \cdots \mid \mathsf{Bool} \mid [\tau] \\ \text{Terms:} & t := \ \cdots \mid \mathsf{True} \mid \mathsf{False} \mid [\,]_\tau \mid t : t \mid \textbf{case } t \textbf{ of } \{\cdots\} \end{array}$$

$$\Gamma \vdash \mathsf{True} : \mathsf{Bool} \quad , \quad \Gamma \vdash \mathsf{False} : \mathsf{Bool} \quad , \quad \Gamma \vdash [\,]_\tau : [\tau]$$

$$\frac{\Gamma \vdash t : \tau \qquad \Gamma \vdash u : [\tau]}{\Gamma \vdash (t : u) : [\tau]}$$

$$\frac{\Gamma \vdash t : \mathsf{Bool} \qquad \Gamma \vdash u : \tau \qquad \Gamma \vdash v : \tau}{\Gamma \vdash (\textbf{case } t \textbf{ of } \{\mathsf{True} \to u\,; \mathsf{False} \to v\}) : \tau}$$

$$\frac{\Gamma \vdash t : [\tau'] \qquad \Gamma \vdash u : \tau \qquad \Gamma, x_1 : \tau', x_2 : [\tau'] \vdash v : \tau}{\Gamma \vdash (\textbf{case } t \textbf{ of } \{[] \to u\,; (x_1 : x_2) \to v\}) : \tau}$$

With the straightforward extension of the semantics and with

$$\begin{array}{rcl} \Delta_{\mathsf{Bool},\rho} & = & \{(\mathsf{True},\mathsf{True}),(\mathsf{False},\mathsf{False})\} \\ \Delta_{[\tau],\rho} & = & \{([x_1,\ldots,x_n],[y_1,\ldots,y_n]) \mid n \geq 0, (x_i,y_i) \in \Delta_{\tau,\rho}\}\,, \end{array}$$

the parametricity theorem still holds.

Now Formal Counterpart to Intuitive Reasoning

Given g of type $\forall\alpha.\ (\alpha \to \mathsf{Bool}) \to ([\alpha] \to [\alpha])$,
by the parametricity theorem:

$$(g, g) \in \Delta_{\forall\alpha.\ (\alpha\to\mathsf{Bool})\to([\alpha]\to[\alpha]),\emptyset}$$

Now Formal Counterpart to Intuitive Reasoning

Given g of type $\forall\alpha.\,(\alpha \to \mathsf{Bool}) \to ([\alpha] \to [\alpha])$,
by the parametricity theorem:

$$
\begin{aligned}
& (\mathrm{g}, \mathrm{g}) \in \Delta_{\forall\alpha.\,(\alpha\to\mathsf{Bool})\to([\alpha]\to[\alpha]),\emptyset} \\
\Leftrightarrow\ & \forall \mathcal{R} \in \mathit{Rel}.\ (\mathrm{g}, \mathrm{g}) \in \Delta_{(\alpha\to\mathsf{Bool})\to([\alpha]\to[\alpha]),[\alpha\mapsto\mathcal{R}]} \\
& \text{by definition of } \Delta
\end{aligned}
$$

Now Formal Counterpart to Intuitive Reasoning

Given g of type $\forall\alpha.\ (\alpha \to \mathsf{Bool}) \to ([\alpha] \to [\alpha])$,
by the parametricity theorem:

$$(\mathrm{g}, \mathrm{g}) \in \Delta_{\forall\alpha.\,(\alpha\to\mathsf{Bool})\to([\alpha]\to[\alpha]),\emptyset}$$
$$\Leftrightarrow \forall\mathcal{R} \in \mathit{Rel}.\ (\mathrm{g}, \mathrm{g}) \in \Delta_{(\alpha\to\mathsf{Bool})\to([\alpha]\to[\alpha]),[\alpha\mapsto\mathcal{R}]}$$
$$\Leftrightarrow \forall\mathcal{R} \in \mathit{Rel}, (a_1, a_2) \in \Delta_{\alpha\to\mathsf{Bool},[\alpha\mapsto\mathcal{R}]}.\ (\mathrm{g}\ a_1, \mathrm{g}\ a_2) \in \Delta_{[\alpha]\to[\alpha],[\alpha\mapsto\mathcal{R}]}$$

by definition of Δ

Now Formal Counterpart to Intuitive Reasoning

Given g of type $\forall\alpha.\ (\alpha \to \mathsf{Bool}) \to ([\alpha] \to [\alpha])$,
by the parametricity theorem:

$$
\begin{aligned}
& (\mathrm{g}, \mathrm{g}) \in \Delta_{\forall\alpha.\,(\alpha\to\mathsf{Bool})\to([\alpha]\to[\alpha]),\emptyset} \\
\Leftrightarrow\ & \forall\mathcal{R} \in \mathit{Rel}.\ (\mathrm{g}, \mathrm{g}) \in \Delta_{(\alpha\to\mathsf{Bool})\to([\alpha]\to[\alpha]),[\alpha\mapsto\mathcal{R}]} \\
\Leftrightarrow\ & \forall\mathcal{R} \in \mathit{Rel}, (a_1, a_2) \in \Delta_{\alpha\to\mathsf{Bool},[\alpha\mapsto\mathcal{R}]}.\ (\mathrm{g}\ a_1, \mathrm{g}\ a_2) \in \Delta_{[\alpha]\to[\alpha],[\alpha\mapsto\mathcal{R}]} \\
\Leftrightarrow\ & \forall\mathcal{R} \in \mathit{Rel}, (a_1, a_2) \in \Delta_{\alpha\to\mathsf{Bool},[\alpha\mapsto\mathcal{R}]}, (l_1, l_2) \in \Delta_{[\alpha],[\alpha\mapsto\mathcal{R}]}. \\
& \quad (\mathrm{g}\ a_1\ l_1, \mathrm{g}\ a_2\ l_2) \in \Delta_{[\alpha],[\alpha\mapsto\mathcal{R}]}
\end{aligned}
$$

by definition of Δ

Now Formal Counterpart to Intuitive Reasoning

Given g of type $\forall\alpha.\ (\alpha \to \mathsf{Bool}) \to ([\alpha] \to [\alpha])$,
by the parametricity theorem:

$$
\begin{aligned}
& (\mathrm{g},\mathrm{g}) \in \Delta_{\forall\alpha.\,(\alpha\to\mathsf{Bool})\to([\alpha]\to[\alpha]),\emptyset} \\
\Leftrightarrow\ & \forall\mathcal{R} \in \mathit{Rel}.\ (\mathrm{g},\mathrm{g}) \in \Delta_{(\alpha\to\mathsf{Bool})\to([\alpha]\to[\alpha]),[\alpha\mapsto\mathcal{R}]} \\
\Leftrightarrow\ & \forall\mathcal{R} \in \mathit{Rel}, (a_1,a_2) \in \Delta_{\alpha\to\mathsf{Bool},[\alpha\mapsto\mathcal{R}]}.\ (\mathrm{g}\ a_1,\mathrm{g}\ a_2) \in \Delta_{[\alpha]\to[\alpha],[\alpha\mapsto\mathcal{R}]} \\
\Leftrightarrow\ & \forall\mathcal{R} \in \mathit{Rel}, (a_1,a_2) \in \Delta_{\alpha\to\mathsf{Bool},[\alpha\mapsto\mathcal{R}]}, (l_1,l_2) \in \Delta_{[\alpha],[\alpha\mapsto\mathcal{R}]}. \\
& \quad (\mathrm{g}\ a_1\ l_1,\mathrm{g}\ a_2\ l_2) \in \Delta_{[\alpha],[\alpha\mapsto\mathcal{R}]} \\
\Rightarrow\ & \forall(a_1,a_2) \in \Delta_{\alpha\to\mathsf{Bool},[\alpha\mapsto f]}, (l_1,l_2) \in (\mathtt{map}\ f). \\
& \quad (\mathrm{g}\ a_1\ l_1,\mathrm{g}\ a_2\ l_2) \in (\mathtt{map}\ f)
\end{aligned}
$$

by instantiating $\mathcal{R} = f$ and realising that $\Delta_{[\alpha],[\alpha\mapsto f]} = \mathtt{map}\ f$

for every function f

Now Formal Counterpart to Intuitive Reasoning

Given $\mathtt{g}$ of type $\forall\alpha.\ (\alpha \to \mathsf{Bool}) \to ([\alpha] \to [\alpha])$,
by the parametricity theorem:

$$
\begin{aligned}
& (\mathtt{g}, \mathtt{g}) \in \Delta_{\forall\alpha.\,(\alpha\to\mathsf{Bool})\to([\alpha]\to[\alpha]),\emptyset} \\
\Leftrightarrow\ & \forall\mathcal{R} \in \mathit{Rel}.\ (\mathtt{g}, \mathtt{g}) \in \Delta_{(\alpha\to\mathsf{Bool})\to([\alpha]\to[\alpha]),[\alpha\mapsto\mathcal{R}]} \\
\Leftrightarrow\ & \forall\mathcal{R} \in \mathit{Rel}, (a_1, a_2) \in \Delta_{\alpha\to\mathsf{Bool},[\alpha\mapsto\mathcal{R}]}.\ (\mathtt{g}\ a_1, \mathtt{g}\ a_2) \in \Delta_{[\alpha]\to[\alpha],[\alpha\mapsto\mathcal{R}]} \\
\Leftrightarrow\ & \forall\mathcal{R} \in \mathit{Rel}, (a_1, a_2) \in \Delta_{\alpha\to\mathsf{Bool},[\alpha\mapsto\mathcal{R}]}, (l_1, l_2) \in \Delta_{[\alpha],[\alpha\mapsto\mathcal{R}]}. \\
& \quad (\mathtt{g}\ a_1\ l_1, \mathtt{g}\ a_2\ l_2) \in \Delta_{[\alpha],[\alpha\mapsto\mathcal{R}]} \\
\Rightarrow\ & \forall(a_1, a_2) \in \Delta_{\alpha\to\mathsf{Bool},[\alpha\mapsto f]}, (l_1, l_2) \in (\mathtt{map}\ f). \\
& \quad (\mathtt{g}\ a_1\ l_1, \mathtt{g}\ a_2\ l_2) \in (\mathtt{map}\ f) \\
\Rightarrow\ & \forall(l_1, l_2) \in (\mathtt{map}\ f).\ (\mathtt{g}\ (p \circ f)\ l_1, \mathtt{g}\ p\ l_2) \in (\mathtt{map}\ f) \\
& \text{by instantiating } (a_1, a_2) = (p \circ f, p) \in \Delta_{\alpha\to\mathsf{Bool},[\alpha\mapsto f]}
\end{aligned}
$$

for every function f and predicate p.

Now Formal Counterpart to Intuitive Reasoning

Given g of type $\forall\alpha.\ (\alpha \to \text{Bool}) \to ([\alpha] \to [\alpha])$,
by the parametricity theorem:

$$
\begin{aligned}
& (\mathrm{g}, \mathrm{g}) \in \Delta_{\forall\alpha.\,(\alpha\to\text{Bool})\to([\alpha]\to[\alpha]),\emptyset} \\
\Leftrightarrow\ & \forall \mathcal{R} \in \mathit{Rel}.\ (\mathrm{g}, \mathrm{g}) \in \Delta_{(\alpha\to\text{Bool})\to([\alpha]\to[\alpha]),[\alpha\mapsto\mathcal{R}]} \\
\Leftrightarrow\ & \forall \mathcal{R} \in \mathit{Rel}, (a_1, a_2) \in \Delta_{\alpha\to\text{Bool},[\alpha\mapsto\mathcal{R}]}.\ (\mathrm{g}\ a_1, \mathrm{g}\ a_2) \in \Delta_{[\alpha]\to[\alpha],[\alpha\mapsto\mathcal{R}]} \\
\Leftrightarrow\ & \forall \mathcal{R} \in \mathit{Rel}, (a_1, a_2) \in \Delta_{\alpha\to\text{Bool},[\alpha\mapsto\mathcal{R}]}, (l_1, l_2) \in \Delta_{[\alpha],[\alpha\mapsto\mathcal{R}]}. \\
& \quad (\mathrm{g}\ a_1\ l_1, \mathrm{g}\ a_2\ l_2) \in \Delta_{[\alpha],[\alpha\mapsto\mathcal{R}]} \\
\Rightarrow\ & \forall (a_1, a_2) \in \Delta_{\alpha\to\text{Bool},[\alpha\mapsto f]}, (l_1, l_2) \in (\mathtt{map}\ f). \\
& \quad (\mathrm{g}\ a_1\ l_1, \mathrm{g}\ a_2\ l_2) \in (\mathtt{map}\ f) \\
\Rightarrow\ & \forall (l_1, l_2) \in (\mathtt{map}\ f).\ (\mathrm{g}\ (p \circ f)\ l_1, \mathrm{g}\ p\ l_2) \in (\mathtt{map}\ f) \\
\Leftrightarrow\ & \forall l_1.\ \mathtt{map}\ f\ (\mathrm{g}\ (p \circ f)\ l_1) = \mathrm{g}\ p\ (\mathtt{map}\ f\ l_1)
\end{aligned}
$$

by inlining

for every function f and predicate p.

Now Formal Counterpart to Intuitive Reasoning

Given $\mathtt{g}$ of type $\forall\alpha.\ (\alpha \to \mathsf{Bool}) \to ([\alpha] \to [\alpha])$,
by the parametricity theorem:

$$
\begin{aligned}
& (\mathtt{g}, \mathtt{g}) \in \Delta_{\forall\alpha.\,(\alpha\to\mathsf{Bool})\to([\alpha]\to[\alpha]),\emptyset} \\
\Leftrightarrow\ & \forall \mathcal{R} \in \mathit{Rel}.\ (\mathtt{g}, \mathtt{g}) \in \Delta_{(\alpha\to\mathsf{Bool})\to([\alpha]\to[\alpha]),[\alpha\mapsto\mathcal{R}]} \\
\Leftrightarrow\ & \forall \mathcal{R} \in \mathit{Rel}, (a_1, a_2) \in \Delta_{\alpha\to\mathsf{Bool},[\alpha\mapsto\mathcal{R}]}.\ (\mathtt{g}\ a_1, \mathtt{g}\ a_2) \in \Delta_{[\alpha]\to[\alpha],[\alpha\mapsto\mathcal{R}]} \\
\Leftrightarrow\ & \forall \mathcal{R} \in \mathit{Rel}, (a_1, a_2) \in \Delta_{\alpha\to\mathsf{Bool},[\alpha\mapsto\mathcal{R}]}, (l_1, l_2) \in \Delta_{[\alpha],[\alpha\mapsto\mathcal{R}]}. \\
& \quad (\mathtt{g}\ a_1\ l_1, \mathtt{g}\ a_2\ l_2) \in \Delta_{[\alpha],[\alpha\mapsto\mathcal{R}]} \\
\Rightarrow\ & \forall (a_1, a_2) \in \Delta_{\alpha\to\mathsf{Bool},[\alpha\mapsto f]}, (l_1, l_2) \in (\mathtt{map}\ f). \\
& \quad (\mathtt{g}\ a_1\ l_1, \mathtt{g}\ a_2\ l_2) \in (\mathtt{map}\ f) \\
\Rightarrow\ & \forall (l_1, l_2) \in (\mathtt{map}\ f).\ (\mathtt{g}\ (p \circ f)\ l_1, \mathtt{g}\ p\ l_2) \in (\mathtt{map}\ f) \\
\Leftrightarrow\ & \forall l_1.\ \mathtt{map}\ f\ (\mathtt{g}\ (p \circ f)\ l_1) = \mathtt{g}\ p\ (\mathtt{map}\ f\ l_1)
\end{aligned}
$$

for every function f and predicate p.

That is what was claimed!

References

J.-Y. Girard.
Interprétation functionelle et élimination des coupures dans l'arithmétique d'ordre supérieure.
PhD thesis, Université Paris VII, 1972.

J.C. Reynolds.
Towards a theory of type structure.
In *Colloque sur la Programmation, Proceedings*, pages 408–423. Springer-Verlag, 1974.

J.C. Reynolds.
Types, abstraction and parametric polymorphism.
In *Information Processing, Proceedings*, pages 513–523. Elsevier Science Publishers B.V., 1983.

P. Wadler.
Theorems for free!
In *Functional Programming Languages and Computer Architecture, Proceedings*, pages 347–359. ACM Press, 1989.