Seminar Presentation

Theorems & Proofs for Free

Sonia
Radboud

January 21, 2026

Overview

1. Overview

2. Theorems For Free

3. Proofs for Free

My Two Papers

- 1st Paper: Theorems for Free! by Philip Wadler (1989)
- 2nd Paper: Proofs for free: Parametricity for dependent types (2012) by Bernardy, Paterson & Jansson

Initial Motivation

We like to generalize

Theorems for Free! by Philip Wadler (1989)

How to derive theorems from parametricity!

Proofs for Free! (2012)

Parametricity and the Curry-Howard correspondence between Pure Type Systems

What can we use parametricity for?

What can we use parametricity for?

- To change our representation

What can we use parametricity for?

- To change our representation
- To go abstract

What can we use parametricity for?

- To change our representation
- To go abstract
- To derive theorems in a more generalized setting

What can we use parametricity for?

For Reynolds, he called it both *Representation theorem* and *Abstraction Theorem*

Theorems for Free! by Philip Wadler (1989)

How to derive theorems from parametricity!

More Motivation

More Motivation

Wadler writes:

> *I co-authored a paper [...], of the nine theorems, five follow immediately [from parametricity]*

Theorems for Free! by Philip Wadler (1989)

What is parametricity?

Theorems for Free! by Philip Wadler (1989)

What is parametricity?

And how does it rely on System F?

System F

First, what is system F?

System F

Also known as

$\lambda 2$ type theory, second-order lambda calculus, polymorhic lambda calculus

System F

Allows for universal quantification over types

System F

Allows for universal quantification over types

$$\forall X.T$$

System F

Allows for universal quantification over types

$$\forall X.T$$

Examples

$$\forall X.\forall Y.X \to Y \to X$$

System F

Types $T ::= X \mid T \to U \mid \forall X.T$
Terms $t ::= x \mid \lambda x : U.\, t \mid t\, u \mid \Lambda X.t \mid t\;{}_U$

System F

- Terms beginning with a Λ are called polymorphic

System F

- Terms beginning with a Λ are called polymorphic
- Λ takes in a **Type Variable**.

System F

- Terms beginning with a Λ are called polymorphic
- Λ takes in a **Type Variable**.
- λ instead takes in a individual variable

System F

- Terms beginning with a Λ are called polymorphic
- Λ takes in a **Type Variable**.
- λ instead takes in a individual variable

Examples

$$\Lambda X.\Lambda Y.\lambda x.\lambda y.x$$

System F

- Terms beginning with a Λ are called polymorphic
- Λ takes in a **Type Variable**.
- λ instead takes in a individual variable

Examples

$$\Lambda X.\Lambda Y.\lambda x.\lambda y.x : \forall X.\forall Y.X \rightarrow Y \rightarrow X$$

Parametricity

The Parametricity Theorem depends on polymorphism

Parametricity

The Parametricity Theorem depends on polymorphism. **Why?**

What is parametricity?

Parametricity allows for theorems to be derived from types only

What is parametricity?

If we derive a theorem for a type of a polymorphic function,
this theorem will hold for every function of that same type

What is parametricity?

We must be in $\lambda 2$ to have polymorphism

What is parametricity?

We must be in $\lambda 2$ to have polymorphism

In fact, we must be in $\lambda 2$ or higher !

Parametricity

Examples

Let $r : \forall X.X^* \to X^*$ be a term of the type *Rearrangement*

Parametricity

Examples

Let $r : \forall X.X^* \to X^*$ be a term of the type *Rearrangement*
We can derive the theorem $a^* \circ r_A = r_{A'} \circ a^*$

Parametricity

Examples

Let $r : \forall X.X^* \to X^*$ be a term of the type *Rearrangement*
We can derive the theorem $a^* \circ r_A = r_{A'} \circ a^*$

Applying a map a to each element of a list and then rearranging

Parametricity

Examples

Let $r : \forall X.X^* \to X^*$ be a term of the type *Rearrangement*
We can derive the theorem $a^* \circ r_A = r_{A'} \circ a^*$

Applying a map a to each element of a list and then rearranging
$=$ rearranging and then applying a map a to each element

Naive Set-Theoretic Parametricity

- There exists no set theoretic model for System F

Naive Set-Theoretic Parametricity

- There exists no set theoretic model for System F
- This was proven by Reynolds

Naive Set-Theoretic Parametricity

- There exists no set theoretic model for System F
- This was proven by Reynolds
- Yet, a naive set-theoretic notation gives intuition

Naive Set-Theoretic Parametricity

- There exists no set theoretic model for System F
- This was proven by Reynolds
- Yet, a naive set-theoretic notation gives intuition
- Types are sets, functions are set-theoretic functions, etc.

Naive Set-Theoretic Parametricity

- There exists no set theoretic model for System F
- This was proven by Reynolds
- Yet, a naive set-theoretic notation gives intuition
- Types are sets, functions are set-theoretic functions, etc.

Examples

If A, B are sets, then $A \to B$ is the set of functions from set A to set B

Naive Set-Theoretic Parametricity

Key idea: To read types as relations!

Naive Set-Theoretic Parametricity

Key idea: To read types as relations!

Let A and A' be sets.

Naive Set-Theoretic Parametricity

Key idea: To read types as relations!

Let A and A' be sets. Let $\mathcal{A}$ be a relation between A and A'.

Naive Set-Theoretic Parametricity

Key idea: To read types as relations!

Let A and A' be sets. Let $\mathcal{A}$ be a relation between A and A'.
We write $(x, x') \in \mathcal{A}$ if $x \in A$ and $x' \in A'$,

Naive Set-Theoretic Parametricity

Key idea: To read types as relations!

Let A and A' be sets. Let $\mathcal{A}$ be a relation between A and A'.
We write $(x, x') \in \mathcal{A}$ if $x \in A$ and $x' \in A'$, and consider x and x' related by $\mathcal{A}$.

Naive Set-Theoretic Parametricity

Key idea: To read types as relations!

Let A and A' be sets. Let $\mathcal{A}$ be a relation between A and A'.
We write $(x, x') \in \mathcal{A}$ if $x \in A$ and $x' \in A'$, and consider x and x' related by $\mathcal{A}$.

Examples

The list relation $\mathcal{A}^* : A^* \Leftrightarrow A'^*$

Naive Set-Theoretic Parametricity

Key idea: To read types as relations!

Let A and A' be sets. Let $\mathcal{A}$ be a relation between A and A'.
We write $(x, x') \in \mathcal{A}$ if $x \in A$ and $x' \in A'$, and consider x and x' related by $\mathcal{A}$.

Examples

The list relation $\mathcal{A}^* : A^* \Leftrightarrow A'^*$
$([x_1, ...x_n], [x_1^{'}, ...x_n^{'}]) \in \mathcal{A}^* \Leftrightarrow (x_1, x_1^{'},) \in \mathcal{A}$ and ... and $(x_n, x_n^{'},) \in \mathcal{A}$

Naive Set-Theoretic Parametricity

Key idea: To read types as relations!

Let A and A' be sets. Let $\mathcal{A}$ be a relation between A and A'.
We write $(x, x') \in \mathcal{A}$ if $x \in A$ and $x' \in A'$, and consider x and x' related by $\mathcal{A}$.

Examples

The list relation $\mathcal{A}^* : A^* \Leftrightarrow A'^*$
$([x_1, ...x_n], [x_1^{'}, ...x_n^{'}]) \in \mathcal{A}^* \Leftrightarrow (x_1, x_1^{'},) \in \mathcal{A}$ and ... and $(x_n, x_n^{'},) \in \mathcal{A}$
i.e. lists are related iff they have the same length and corresponding elements are related.

Parametricity Proposition

Parametricity Proposition

If t is a term of type T and $\mathcal{T}$ is the relation corresponding to the type T, then $(t, t) \in \mathcal{T}$.

Parametricity Proposition

Parametricity Proposition

If t is a term of type T and $\mathcal{T}$ is the relation corresponding to the type T, then $(t, t) \in \mathcal{T}$.

Examples

$$r : \forall X : X^* \to X^*$$

Parametricity Proposition

Parametricity Proposition

If t is a term of type T and $\mathcal{T}$ is the relation corresponding to the type T, then $(t, t) \in \mathcal{T}$.

Examples

$$r : \forall X : X^* \to X^* \qquad \Rightarrow \qquad (r, r) \in \forall \mathcal{X}.\mathcal{X}^* \to \mathcal{X}^*$$

Parametricity under a naive set-theoretic model

Interpret $\rightarrow$ and $\forall$ as relations

Interpret ∀ as an operation on relation

Polymorphic functions are related if they take related types into related results

Interpret ∀ as an operation on relation

Polymorphic functions are related if they take related types into related results

$$(g, g') \in \forall \mathcal{X}.\mathcal{F}(\mathcal{X}) \Leftrightarrow \text{for all } \mathcal{A},\ (g_A, g'_{A'}) \in \mathcal{F}(\mathcal{A})$$

Interpret $\forall$ as an operation on relation

Polymorphic functions are related if they take related types into related results

$$(g, g') \in \forall \mathcal{X}.\mathcal{F}(\mathcal{X}) \Leftrightarrow \text{for all } \mathcal{A},\ (g_A, g'_{A'}) \in \mathcal{F}(\mathcal{A})$$

Examples

$$(r, r) \in \forall \mathcal{X}.\mathcal{X}^* \to \mathcal{X}^*$$

Interpret ∀ as an operation on relation

Polymorphic functions are related if they take related types into related results

$$(g, g') \in \forall \mathcal{X}.\mathcal{F}(\mathcal{X}) \Leftrightarrow \text{for all } \mathcal{A},\ (g_A, g'_{A'}) \in \mathcal{F}(\mathcal{A})$$

Examples

$(r, r) \in \forall \mathcal{X}.\mathcal{X}^* \rightarrow \mathcal{X}^*$ $\Rightarrow$ for all $\mathcal{A}, (r_A, r_{A'}) \in \mathcal{A}^* \rightarrow \mathcal{A}^*$

Interpret $\rightarrow$ as a relation

Functions are related if they take related arguments into related results

Interpret $\rightarrow$ as a relation

Functions are related if they take related arguments into related results

$$(f, f') \in \mathcal{A} \rightarrow \mathcal{B} \Leftrightarrow \text{for all } (x, x') \in \mathcal{A}\ ,\ (f\ x, f'\ x') \in \mathcal{B}$$

Interpret $\rightarrow$ as a relation

Functions are related if they take related arguments into related results

$$(f, f') \in \mathcal{A} \rightarrow \mathcal{B} \Leftrightarrow \text{for all } (x, x') \in \mathcal{A}\ ,\ (f\,x, f'\,x') \in \mathcal{B}$$

Examples

for all $\mathcal{A}$,
$(\boldsymbol{r_A}, \boldsymbol{r_{A'}}) \in \boldsymbol{\mathcal{A}^*} \rightarrow \boldsymbol{\mathcal{A}^*}$

for all $\mathcal{A}$,

Interpret $\rightarrow$ as a relation

Functions are related if they take related arguments into related results

$$(f, f') \in \mathcal{A} \rightarrow \mathcal{B} \Leftrightarrow \text{for all } (x, x') \in \mathcal{A}\ , (f\ x, f'\ x') \in \mathcal{B}$$

Examples

$$\begin{array}{c} \text{for all } \mathcal{A}, \\ (\boldsymbol{r_A}, \boldsymbol{r_{A'}}) \in \boldsymbol{\mathcal{A}^*} \rightarrow \boldsymbol{\mathcal{A}^*} \end{array} \quad \Rightarrow \quad \begin{array}{c} \text{for all } \mathcal{A}, \\ \textbf{for all } (\boldsymbol{x}, \boldsymbol{x'}) \in \boldsymbol{\mathcal{A}^*}, (\boldsymbol{r_A}\ \boldsymbol{x}, \boldsymbol{r_{A'}}\ \boldsymbol{x'}) \in \boldsymbol{\mathcal{A}^*} \end{array}$$

Example for Rearrangements

Examples

for all $\mathcal{A}$, for all $(x, x') \in \boldsymbol{\mathcal{A}}^*, (r_A\, x, r_{A'}\, x') \in \boldsymbol{\mathcal{A}}^*$

Example for Rearrangements

Examples

for all $\mathcal{A}$, for all $(x, x') \in \boldsymbol{\mathcal{A}}^*, (r_A\, x, r_{A'}\, x') \in \boldsymbol{\mathcal{A}}^*$

Specializing relation $\mathcal{A}$ as a function $a : A \to A'$,

Example for Rearrangements

Examples

for all $\mathcal{A}$, for all $(x, x') \in \boldsymbol{\mathcal{A}}^*, (r_A\, x, r_{A'}\, x') \in \boldsymbol{\mathcal{A}}^*$

Specializing relation $\mathcal{A}$ as a function $a : A \to A'$,we get
1) for all $(x, x') \in \mathcal{A}^*$

Example for Rearrangements

Examples

for all $\mathcal{A}$, for all $(x, x') \in \boldsymbol{\mathcal{A}}^*, (r_A\, x, r_{A'}\, x') \in \boldsymbol{\mathcal{A}}^*$

Specializing relation $\mathcal{A}$ as a function $a : A \to A'$,we get
1) for all $(x, x') \in \mathcal{A}^* \Rightarrow a^* x = x'$

Example for Rearrangements

Examples

for all $\mathcal{A}$, for all $(x, x') \in \boldsymbol{\mathcal{A}^*}, (r_A\, x, r_{A'}\, x') \in \boldsymbol{\mathcal{A}^*}$

Specializing relation $\mathcal{A}$ as a function $a : A \rightarrow A'$,we get
1) for all $(x, x') \in \mathcal{A}^* \Rightarrow a^* x = x'$
2) for all $(r_A\, x, r_{A'}\, x') \in \mathcal{A}^*$

Example for Rearrangements

Examples

for all $\mathcal{A}$, for all $(x, x') \in \boldsymbol{\mathcal{A}}^*, (r_A\, x, r_{A'}\, x') \in \boldsymbol{\mathcal{A}}^*$

Specializing relation $\mathcal{A}$ as a function $a : A \to A'$,we get
1) for all $(x, x') \in \mathcal{A}^* \Rightarrow a^* x = x'$
2) for all $(r_A\, x, r_{A'}\, x') \in \mathcal{A}^* \Rightarrow a^*(r_A\, x) = r_{A'}\, x'$

Example for Rearrangements

Examples

for all $\mathcal{A}$, for all $(x, x') \in \boldsymbol{\mathcal{A}}^*, (r_A\, x, r_{A'}\, x') \in \boldsymbol{\mathcal{A}}^*$

Specializing relation $\mathcal{A}$ as a function $a : A \to A'$,we get
1) for all $(x, x') \in \mathcal{A}^* \Rightarrow a^* x = x'$
2) for all $(r_A\, x, r_{A'}\, x') \in \mathcal{A}^* \Rightarrow a^*(r_A\, x) = r_{A'}\, x'$

$a^*\,(r_A\, x) = r_{A'}\, x'$ from 2)

Example for Rearrangements

Examples

for all $\mathcal{A}$, for all $(x, x') \in \boldsymbol{\mathcal{A}^*}, (r_A\, x, r_{A'}\, x') \in \boldsymbol{\mathcal{A}^*}$

Specializing relation $\mathcal{A}$ as a function $a : A \to A'$,we get
1) for all $(x, x') \in \mathcal{A}^* \Rightarrow a^* x = x'$
2) for all $(r_A\, x, r_{A'}\, x') \in \mathcal{A}^* \Rightarrow a^*(r_A\, x) = r_{A'}\, x'$

$a^*\,(r_A\, x) = r_{A'}\, x'$ from 2)
$a^*\,(r_A\, x) = r_{A'}\, \boldsymbol{x'}$

Example for Rearrangements

Examples

for all $\mathcal{A}$, for all $(x, x') \in \boldsymbol{\mathcal{A}}^*, (r_A\, x, r_{A'}\, x') \in \boldsymbol{\mathcal{A}}^*$

Specializing relation $\mathcal{A}$ as a function $a : A \to A'$,we get
1) for all $(x, x') \in \mathcal{A}^* \Rightarrow a^* x = x'$
2) for all $(r_A\, x, r_{A'}\, x') \in \mathcal{A}^* \Rightarrow a^*(r_A\, x) = r_{A'}\, x'$

$a^*\,(r_A\, x) = r_{A'}\, x'$ from 2)
$a^*\,(r_A\, x) = r_{A'}\, \boldsymbol{x'}$
$a^*\,(r_A\, x) = r_{A'}\, (\boldsymbol{a^*\, x})$ use 1)

Example for Rearrangements

Examples

for all $\mathcal{A}$, for all $(x, x') \in \boldsymbol{\mathcal{A}}^*, (r_A\, x, r_{A'}\, x') \in \boldsymbol{\mathcal{A}}^*$

Specializing relation $\mathcal{A}$ as a function $a : A \to A'$,we get
1) for all $(x, x') \in \mathcal{A}^* \Rightarrow a^* x = x'$
2) for all $(r_A\, x, r_{A'}\, x') \in \mathcal{A}^* \Rightarrow a^*(r_A\, x) = r_{A'}\, x'$

$a^*\,(r_A\, x) = r_{A'}\, x'$ from 2)
$a^*\,(r_A\, x) = r_{A'}\, \boldsymbol{x'}$
$a^*\,(r_A\, x) = r_{A'}\, (\boldsymbol{a^*\, x})$ use 1) $\Rightarrow a^* \circ r_A = r_{A'} \circ a^*$

Example

Examples

Let $r : \forall X.X^* \to X^*$ be a term of the type *Rearrangement*
We can derive the theorem, for $a : A \to A'$, $a^* \circ r_A = r_{A'} \circ a^*$

Example

Examples

Let $r : \forall X.X^* \to X^*$ be a term of the type *Rearrangement*
We can derive the theorem, for $a : A \to A'$, $a^* \circ r_A = r_{A'} \circ a^*$

Applying a map a to each element of a list and then rearranging
$=$ rearranging and then applying a map a to each element

Generalization

Examples

Let $t : T$ be a term of a specific type
We can derive a theorem $\ldots t \ldots = \ldots t \ldots$

Generalization

Examples

Let $t : T$ be a term of a specific type

We can derive a theorem $\ldots t \ldots = \ldots t \ldots$

Then this theorem then holds for all terms t of type T

More examples

Assume $a : A \to A'$ and $b : B \to B'$.

$$head : \forall X.\ X^* \to X$$
$$a \circ head_A = head_{A'} \circ a^*$$

$$tail : \forall X.\ X^* \to X^*$$
$$a^* \circ tail_A = tail_{A'} \circ a^*$$

$$(+\!\!+) : \forall X.\ X^* \to X^* \to X^*$$
$$a^*\ (xs +\!\!+_A ys) = (a^*\ xs) +\!\!+_{A'} (a^*\ ys)$$

$$concat : \forall X.\ X^{**} \to X^*$$
$$a^* \circ concat_A = concat_{A'} \circ a^{**}$$

$$fst : \forall X.\ \forall Y.\ X \times Y \to X$$
$$a \circ fst_{AB} = fst_{A'B'} \circ (a \times b)$$

$$snd : \forall X.\ \forall Y.\ X \times Y \to Y$$
$$b \circ snd_{AB} = snd_{A'B'} \circ (a \times b)$$

$$zip : \forall X.\ \forall Y.\ (X^* \times Y^*) \to (X \times Y)^*$$
$$(a \times b)^* \circ zip_{AB} = zip_{A'B'} \circ (a^* \times b^*)$$

$$filter : \forall X.\ (X \to Bool) \to X^* \to X^*$$
$$a^* \circ filter_A\ (p' \circ a) = filter_{A'}\ p' \circ a^*$$

$$sort : \forall X.\ (X \to X \to Bool) \to X^* \to X^*$$
$$\text{if for all } x, y \in A,\quad (x < y) = (a\ x <'\ a\ y) \text{ then}$$
$$a^* \circ sort_A\ (<) = sort_{A'}\ (<') \circ a^*$$

$$fold : \forall X.\ \forall Y.\ (X \to Y \to Y) \to Y \to X^* \to Y$$
$$\text{if for all } x \in A, y \in B,\quad b\ (x \oplus y) = (a\ x) \otimes (b\ y) \text{ and } b\ u = u' \text{ then}$$
$$b \circ fold_{AB}\ (\oplus)\ u = fold_{A'B'}\ (\otimes)\ u' \circ a^*$$

$$I : \forall X.\ X \to X$$
$$a \circ I_A = I_{A'} \circ a$$

$$K : \forall X.\ \forall Y.\ X \to Y \to X$$
$$a\ (K_{AB}\ x\ y) = K_{A'B'}\ (a\ x)\ (b\ y)$$

Figure 1: Examples of theorems from types

Proofs for Free! (2012)

Parametricity and the Curry-Howard correspondence between Pure Type Systems

Proofs for Free! (2012)

For a Pure Type System used as a programming language,
there is a Pure Type System that can be used as a logic for Parametricity

Proofs for Free

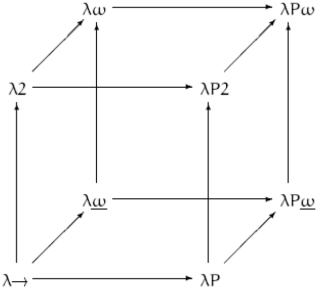

Figure: Pure type systems

Pure Type Systems (PTS)

- $\mathbb{T} = \mathbb{C}$
  $\mathbb{V}$
  $\mathbb{T}\,\mathbb{T}$
  $\lambda\mathbb{V} : \mathbb{T}.\mathbb{T}$
  $\forall\mathbb{V} : \mathbb{T}.\mathbb{T}$

Pure Type Systems (PTS)

- $\mathbb{T} = \mathbb{C}$
  $\mathbb{V}$
  $\mathbb{T}\,\mathbb{T}$
  $\lambda\mathbb{V} : \mathbb{T}.\mathbb{T}$
  $\forall\mathbb{V} : \mathbb{T}.\mathbb{T}$
- Specification $(\mathbb{S}, \mathbb{A}, \mathbb{R})$

Pure Type Systems (PTS)

- $\mathbb{T} = \mathbb{C}$
  $\mathbb{V}$
  $\mathbb{T}\,\mathbb{T}$
  $\lambda\mathbb{V} : \mathbb{T}.\mathbb{T}$
  $\forall\mathbb{V} : \mathbb{T}.\mathbb{T}$
- Specification $(\mathbb{S}, \mathbb{A}, \mathbb{R})$
- $\mathbb{S} \subseteq \mathbb{C}$ sorts
- $\mathbb{A} \subseteq \mathbb{C} \times \mathbb{S}$ axioms
- $\mathbb{R} \subseteq \mathbb{S} \times \mathbb{S} \times \mathbb{S}$ typing rules

Typing rules for PTS

$$\frac{}{\vdash \mathfrak{c} : s}\ \mathfrak{c} : s \in \mathbb{A}$$

AXIOM

$$\frac{\Gamma \vdash A : s}{\Gamma, \mathrm{x} : A \vdash \mathrm{x} : A}$$

START

$$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s}{\Gamma, \mathrm{x} : C \vdash A : B}$$

WEAKING

$$\frac{\Gamma \vdash A : s_1 \qquad \Gamma, \mathrm{x} : A \vdash B : s_2}{\Gamma \vdash (\forall \mathrm{x} : A.B) : s_3}\ (s_1, s_2, s_3) \in \mathbb{R}$$

PRODUCT

$$\frac{\Gamma \vdash F : (\forall \mathrm{x} : A : B) \qquad \Gamma \vdash a : A}{\Gamma \vdash F\, a : B[\mathrm{x} \mapsto a]}$$

APPLICATION

$$\frac{\Gamma, \mathrm{x} : A \vdash b : B \qquad \Gamma \vdash (\forall \mathrm{x} : A : B) : s}{\Gamma \vdash (\lambda \mathrm{x} : A.B) : (\forall \mathrm{x} : A : B)}$$

ABSTRACTION

$$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : s \qquad B =_\beta B'}{\Gamma \vdash A : B'}$$

CONVERSION

The rule (s_1, s_2, s_2) is often written as $s1 \rightsquigarrow s2$.

Family of λ-calculi

- I_ω is a PTS with sort hierarchies
  - $\mathbb{S} = \{*_i | i \in \mathbb{N}\}$
  - $\mathbb{A} = \{*_i : *_{i+1} \in \mathbb{N}\}$
  - $\mathbb{R} = \{(*_i, *_j, *_{max(i,j)}) | i, j \in \mathbb{N}\}$

Family of λ-calculi

- I_ω is a PTS with sort hierarchies
  - $\mathbb{S} = \{*_i | i \in \mathbb{N}\}$
  - $\mathbb{A} = \{*_i : *_{i+1} \in \mathbb{N}\}$
  - $\mathbb{R} = \{(*_i, *_j, *_{max(i,j)}) | i, j \in \mathbb{N}\}$
- CC_ω is a PTS with kind hierarchies
  - $\mathbb{S} = \{*\} \cup \{\Box_i | i \in \mathbb{N}\}$
  - $\mathbb{A} = \{* : \Box_0\} \cup \{\Box_i : \Box_{i+1} | i \in \mathbb{N}\}$
  - $\mathbb{R} = \{* \rightsquigarrow *, * \rightsquigarrow \Box_i, \Box_i \rightsquigarrow * | i \in \mathbb{N}\} \cup \{(\Box_i, \Box_j, \Box_{max(i,j)}) | i, j \in \mathbb{N}\}$

Family of λ-calculi

- I_ω is a PTS with sort hierarchies
  - $\mathbb{S} = \{*_i | i \in \mathbb{N}\}$
  - $\mathbb{A} = \{*_i : *_{i+1} \in \mathbb{N}\}$
  - $\mathbb{R} = \{(*_i, *_j, *_{max(i,j)}) | i, j \in \mathbb{N}\}$
- CC_ω is a PTS with kind hierarchies
  - $\mathbb{S} = \{*\} \cup \{\Box_i | i \in \mathbb{N}\}$
  - $\mathbb{A} = \{* : \Box_0\} \cup \{\Box_i : \Box_{i+1} | i \in \mathbb{N}\}$
  - $\mathbb{R} = \{* \rightsquigarrow *, * \rightsquigarrow \Box_i, \Box_i \rightsquigarrow * | i \in \mathbb{N}\} \cup \{(\Box_i, \Box_j, \Box_{max(i,j)}) | i, j \in \mathbb{N}\}$
- CC$\subseteq$ CC_ω and $I_\omega \subseteq CC_\omega$

Logical Framework

- Types correspond to propositions

Logical Framework

- Types correspond to propositions
- Terms correspond to proofs

Logical Framework

- Types correspond to propositions
- Terms correspond to proofs

Logical Framework

- Source and target PTS

Logical Framework

- Source and target PTS
- This is familiar from Type Theory study group!

Logical Framework

- Source and target PTS
- This is familiar from Type Theory study group!
- Proof Language & Programming Language

Logical Framework

- Source and target PTS
- This is familiar from Type Theory study group!
- Proof Language & Programming Language

Proof Language	$= \lambda C$	$\approx$ Rocq
Programming Language	$= \lambda\omega$	$\approx$ Haskell/Ocaml

Source and Target

SOURCE	Programming Language
TARGET	Proof Language

Source and Target

- The target PTS must include the source PTS

Source and Target

- The target PTS must include the source PTS
- Then all the source terms can be expressed

Reflecting System

The target must *reflect* the source

Reflective

CC_ω reflects each of the systems in the λ-cube

Reflective

CC_ω reflects each of the systems in the λ-cube

CC_ω and I_ω are both self-reflective

Reflective

CC_ω reflects each of the systems in the λ-cube

CC_ω and I_ω are both self-reflective
we can write programs + derive valid statements about them within the same PTS

Translations

$[\![-]\!]$

Translations

$[\![-]\!]$ turns types into relations

Translations

$[\![-]\!]$ turns types into relations and terms into proofs

Function Types ($\lambda \rightarrow$)

λ-**calculus**	$\mathbb{R}$
Simply Typed	$\mathbb{R}_\lambda = \{* \rightsquigarrow *\}$

Function Types ($\lambda \rightarrow$)

λ-**calculus**	$\mathbb{R}$
Simply Typed	$\mathbb{R}_\lambda = \{* \leadsto *\}$

$\mathcal{A} \rightarrow \mathcal{B} : (A \rightarrow B) \Leftrightarrow (A' \rightarrow B')$ is defined by
$(f, f') \in \mathcal{A} \rightarrow \mathcal{B} \Leftrightarrow$ for all $(x, x') \in \mathcal{A}$, $(f\,x, f'\,x') \in \mathcal{B}$
i.e. functions are related if they take related arguments into related results.

Function Types ($\lambda \rightarrow$)

λ-calculus	$\mathbb{R}$
Simply Typed	$\mathbb{R}_\lambda = \{* \rightsquigarrow *\}$

$\mathcal{A} \rightarrow \mathcal{B} : (A \rightarrow B) \Leftrightarrow (A' \rightarrow B')$ is defined by
$(f, f') \in \mathcal{A} \rightarrow \mathcal{B} \Leftrightarrow$ for all $(x, x') \in \mathcal{A}$, $(f\,x, f'\,x') \in \mathcal{B}$
i.e. functions are related if they take related arguments into related results.

$[\![A \rightarrow B]\!] : [\![*]\!]\,(A \rightarrow B)\,(A \rightarrow B)$
$[\![A \rightarrow B]\!]\;\mathsf{f}_1\;\mathsf{f}_2 = \forall \mathsf{a}_1 : A. \forall \mathsf{a}_2 : A.\, [\![A]\!]\;\mathsf{a}_1\;\mathsf{a}_2 \rightarrow [\![B]\!]\,(\mathsf{f}_1\;\mathsf{a}_1)\,(\mathsf{f}_2\;\mathsf{a}_2)$

Type Schemes (System F)

λ-**calculus**	$\mathbb{R}$
Simply Typed	$\mathbb{R}_\lambda = \{* \rightsquigarrow *\}$
System F	$\mathbb{R}_\mathsf{F} = \mathbb{R}_\lambda \cup \{\Box \rightsquigarrow *\}$

Type Schemes (System F)

λ-**calculus**	$\mathbb{R}$
Simply Typed	$\mathbb{R}_\lambda = \{* \leadsto *\}$
System F	$\mathbb{R}_\mathsf{F} = \mathbb{R}_\lambda \cup \{\Box \leadsto *\}$

The relation $\forall \mathcal{X}.\mathcal{F}(\mathcal{X}) : \forall X.F(X) \Leftrightarrow \forall X'.F'(X')$ is defined by
$(g, g') \in \forall \mathcal{X}.\mathcal{F}(\mathcal{X}) \Leftrightarrow$ for all $\mathcal{A} : A \Leftrightarrow A'$, $(g_A, g'_{A'}) \in \mathcal{F}(\mathcal{A})$
I.e. polymorphic functions are related if they take related types into related results.

Type Schemes (System F)

λ-**calculus**	$\mathbb{R}$
Simply Typed	$\mathbb{R}_\lambda = \{* \rightsquigarrow *\}$
System F	$\mathbb{R}_\mathsf{F} = \mathbb{R}_\lambda \cup \{\Box \rightsquigarrow *\}$

The relation $\forall \mathcal{X}.\mathcal{F}(\mathcal{X}) : \forall X.F(X) \Leftrightarrow \forall X'.F'(X')$ is defined by
$(g, g') \in \forall \mathcal{X}.\mathcal{F}(\mathcal{X}) \Leftrightarrow$ for all $\mathcal{A} : A \Leftrightarrow A'$, $(g_A, g'_{A'}) \in \mathcal{F}(\mathcal{A})$
I.e. polymorphic functions are related if they take related types into related results.

$[\![\forall A : *.B]\!] : [\![*]\!]\,(\forall A : *.B)\,(\forall A : *.B)$
$[\![\forall A : *.B]\!]\,\mathsf{g}_1\,\mathsf{g}_2 = \forall \mathsf{A}_1 : *.\forall \mathsf{A}_2 : *.\forall \mathsf{A}_\mathsf{R} : [\![*]\!]\,\mathsf{A}_1\,\mathsf{A}_2.\,[\![B]\!]\,(\mathsf{g}_1\,\mathsf{A}_1)\,(\mathsf{g}_2\,\mathsf{A}_2)$

Type Constructors (System F_ω)

λ-**calculus**	$\mathbb{R}$
Simply Typed	$\mathbb{R}_\lambda = \{* \rightsquigarrow *\}$
System F	$\mathbb{R}_{\mathsf{F}} = \mathbb{R}_\lambda \cup \{\square \rightsquigarrow *\}$
System Fω	$\mathbb{R}_{\mathsf{F}\omega} = \mathbb{R}_{\mathsf{F}} \cup \{\square \rightsquigarrow \square\}$

Type Constructors (System F_ω)

λ-calculus	$\mathbb{R}$
Simply Typed	$\mathbb{R}_\lambda = \{* \rightsquigarrow *\}$
System F	$\mathbb{R}_\mathsf{F} = \mathbb{R}_\lambda \cup \{\Box \rightsquigarrow *\}$
System Fω	$\mathbb{R}_{\mathsf{F}\omega} = \mathbb{R}_\mathsf{F} \cup \{\Box \rightsquigarrow \Box\}$

Types can depend on types

Type Constructors (System F_ω)

λ-**calculus**	$\mathbb{R}$
Simply Typed	$\mathbb{R}_\lambda = \{* \leadsto *\}$
System F	$\mathbb{R}_\mathsf{F} = \mathbb{R}_\lambda \cup \{\Box \leadsto *\}$
System Fω	$\mathbb{R}_{\mathsf{F}\omega} = \mathbb{R}_\mathsf{F} \cup \{\Box \leadsto \Box\}$

Types can depend on types

Types constructors are related iff they take related input types into related output types

Type Constructors (System F_ω)

λ-calculus	$\mathbb{R}$
Simply Typed	$\mathbb{R}_\lambda = \{* \leadsto *\}$
System F	$\mathbb{R}_F = \mathbb{R}_\lambda \cup \{\Box \leadsto *\}$
System Fω	$\mathbb{R}_{F\omega} = \mathbb{R}_F \cup \{\Box \leadsto \Box\}$

Types can depend on types

Types constructors are related iff they take related input types into related output types

$$[\![* \to *]\!] : [\![\Box]\!]\,(* \to *)\,(* \to *)$$
$$[\![* \to *]\!]\,F_1\,F_2 = \forall A_1 : *.\forall A_2 : *.[\![*]\!]\,A_1\,A_2 \to [\![*]\!]\,(F_1\,A_1)\,(F_2\,A_2)$$

Dependent Functions (CC)

λ-**calculus**	$\mathbb{R}$
Simply Typed	$\mathbb{R}_\lambda = \{* \rightsquigarrow *\}$
System F	$\mathbb{R}_\mathsf{F} = \mathbb{R}_\lambda \cup \{\Box \rightsquigarrow *\}$
System Fω	$\mathbb{R}_{\mathsf{F}\omega} = \mathbb{R}_\mathsf{F} \cup \{\Box \rightsquigarrow \Box\}$
Calculus of Constructions (CC)	$\mathbb{R}_{\mathsf{CC}} = \mathbb{R}_{\mathsf{F}\omega} \cup \{* \rightsquigarrow \Box\}$,

Dependent Functions (CC)

λ-**calculus**	$\mathbb{R}$
Simply Typed	$\mathbb{R}_\lambda = \{* \rightsquigarrow *\}$
System F	$\mathbb{R}_\mathsf{F} = \mathbb{R}_\lambda \cup \{\square \rightsquigarrow *\}$
System Fω	$\mathbb{R}_{\mathsf{F}\omega} = \mathbb{R}_\mathsf{F} \cup \{\square \rightsquigarrow \square\}$
Calculus of Constructions (CC)	$\mathbb{R}_{\mathsf{CC}} = \mathbb{R}_{\mathsf{F}\omega} \cup \{* \rightsquigarrow \square\}$,

Types can depend on terms

Dependent Functions (CC)

λ-**calculus**	$\mathbb{R}$
Simply Typed	$\mathbb{R}_\lambda = \{* \rightsquigarrow *\}$
System F	$\mathbb{R}_\mathsf{F} = \mathbb{R}_\lambda \cup \{\Box \rightsquigarrow *\}$
System Fω	$\mathbb{R}_{\mathsf{F}\omega} = \mathbb{R}_\mathsf{F} \cup \{\Box \rightsquigarrow \Box\}$
Calculus of Constructions (CC)	$\mathbb{R}_{\mathsf{CC}} = \mathbb{R}_{\mathsf{F}\omega} \cup \{* \rightsquigarrow \Box\}$,

Types can depend on terms

Dependent functions are related iff they take related value variables into related types

Dependent Functions (CC)

λ-**calculus**	$\mathbb{R}$
Simply Typed	$\mathbb{R}_\lambda = \{* \rightsquigarrow *\}$
System F	$\mathbb{R}_\mathsf{F} = \mathbb{R}_\lambda \cup \{\Box \rightsquigarrow *\}$
System Fω	$\mathbb{R}_{\mathsf{F}\omega} = \mathbb{R}_\mathsf{F} \cup \{\Box \rightsquigarrow \Box\}$
Calculus of Constructions (CC)	$\mathbb{R}_{\mathsf{CC}} = \mathbb{R}_{\mathsf{F}\omega} \cup \{* \rightsquigarrow \Box\}$,

Types can depend on terms

Dependent functions are related iff they take related value variables into related types

$$[\![\forall x : A.B]\!] : [\![*]\!]\,(\forall x : A.B)\,(\forall x : A.B)$$
$$[\![\forall x : A.B]\!]\, f_1\, f_2 = \forall x_1 : A.\forall x_2 : A.\forall x_R : [\![A]\!]\, x_1\, x_2.\, [\![B]\!]\,(f_1\, x_1)\,(f_2\, x_2)$$

λ cube

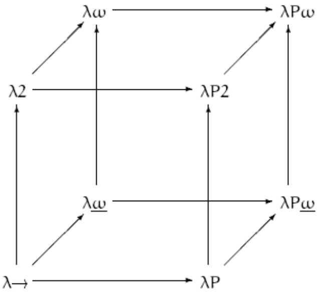

Figure: Pure type systems

Conclusions on Proofs for Free

Conclusions on Proofs for Free

We love to generalize

Conclusions on Proofs for Free

Conclusions on Proofs for Free

- Proofs for Free! expands on the ideas of Theorems for Free!

Conclusions on Proofs for Free

- Proofs for Free! expands on the ideas of Theorems for Free!
- It allows us to consider all the $\lambda-$calculi

Conclusions on Proofs for Free

- Curry-Howard correspondence

Conclusions

- Parametricity allows us to

Conclusions

- Parametricity allows us to
  - Derive theorems that holds for all terms of a given type (Wadler)

Conclusions

- Parametricity allows us to
  - Derive theorems that holds for all terms of a given type (Wadler)
  - Terms evaluated in related environments yield related values (Reynolds)

Conclusions

- Parametricity allows us to
  - Derive theorems that holds for all terms of a given type (Wadler)
  - Terms evaluated in related environments yield related values (Reynolds)
- Using the Curry-Howard correspondence:

Conclusions

- Parametricity allows us to
  - Derive theorems that holds for all terms of a given type (Wadler)
  - Terms evaluated in related environments yield related values (Reynolds)
- Using the Curry-Howard correspondence:

For a PTS used as a programming language,
there is a PTS that can be used as a logic for parametricity

Q&A

Reflecting System

A PTS $S^r = (\mathbb{S}^r, \mathbb{A}^r, \mathbb{R}^r)$ reflects a PTS $S = (\mathbb{S}, \mathbb{A}, \mathbb{R})$ if S is a subsystem of S^r and

- for each sort $s \in \mathbb{S}$,
  - $\mathbb{S}^r$ contains $\tilde{s}, s_1, s_2, s_3$
  - $\mathbb{A}^r$ contains $s : s_1, \tilde{s} : s_2$, and $s_2 : s_3$.
  - $\mathbb{R}^r$ contains $s \rightsquigarrow s_2$ and $s_1 \rightsquigarrow s_3$.
- For each axiom $s : t \in \mathbb{A}, s_2 = \tilde{t}$
- For each rule $(s', s'', s''') \in \mathbb{R}$, $\mathbb{R}^r$ contains rules $(\tilde{s'}, \tilde{s''}, \tilde{s'''})$ and $s' \rightsquigarrow \tilde{s'''}$.

Reflecting System

A PTS $S^r = (\mathbb{S}^r, \mathbb{A}^r, \mathbb{R}^r)$ reflects a PTS $S = (\mathbb{S}, \mathbb{A}, \mathbb{R})$ if S is a subsystem of S^r and

- for each sort $s \in \mathbb{S}$,
  - $\mathbb{S}^r$ contains $\tilde{s}, s_1, s_2, s_3$
  - $\mathbb{A}^r$ contains $s : s_1, \tilde{s} : s_2$, and $s_2 : s_3$.
  - $\mathbb{R}^r$ contains $s \rightsquigarrow s_2$ and $s_1 \rightsquigarrow s_3$.
- For each axiom $s : t \in \mathbb{A}, s_2 = \tilde{t}$
- For each rule $(s', s'', s''') \in \mathbb{R}$, $\mathbb{R}^r$ contains rules $(\tilde{s'}, \tilde{s''}, \tilde{s'''})$ and $s' \rightsquigarrow \tilde{s'''}$.

- CC_ω reflects each of the systems in the λ-cube with $s = \tilde{s}$.

Reflecting System

A PTS $S^r = (\mathbb{S}^r, \mathbb{A}^r, \mathbb{R}^r)$ reflects a PTS $S = (\mathbb{S}, \mathbb{A}, \mathbb{R})$ if S is a subsystem of S^r and

- for each sort $s \in \mathbb{S}$,
  - $\mathbb{S}^r$ contains $\tilde{s}, s_1, s_2, s_3$
  - $\mathbb{A}^r$ contains $s : s_1, \tilde{s} : s_2$, and $s_2 : s_3$.
  - $\mathbb{R}^r$ contains $s \rightsquigarrow s_2$ and $s_1 \rightsquigarrow s_3$.
- For each axiom $s : t \in \mathbb{A}, s_2 = \tilde{t}$
- For each rule $(s', s'', s''') \in \mathbb{R}$, $\mathbb{R}^r$ contains rules $(\tilde{s'}, \tilde{s''}, \tilde{s'''})$ and $s' \rightsquigarrow \tilde{s'''}$.

- CC_ω reflects each of the systems in the λ-cube with $s = \tilde{s}$.
- S is reflective if S reflects itself with $s = \tilde{s}$.

Family of λ-calculi

$$\mathbb{S} = \{*, \Box\} \text{ (types, kinds)}, \ \mathbb{A} = \{* : \Box\}$$

λ-**calculus**	$\mathbb{R}$
Simply Typed	$\mathbb{R}_\lambda = \{* \rightsquigarrow *\}$

Family of λ-calculi

$$\mathbb{S} = \{*, \square\} \text{ (types, kinds)}, \ \mathbb{A} = \{* : \square\}$$

λ-calculus	$\mathbb{R}$
Simply Typed	$\mathbb{R}_\lambda = \{* \rightsquigarrow *\}$
System F	$\mathbb{R}_\mathsf{F} = \mathbb{R}_\lambda \cup \{\square \rightsquigarrow *\}$

Family of λ-calculi

$$\mathbb{S} = \{*, \square\} \text{ (types, kinds)}, \ \mathbb{A} = \{* : \square\}$$

λ-**calculus**	$\mathbb{R}$
Simply Typed	$\mathbb{R}_\lambda = \{* \rightsquigarrow *\}$
System F	$\mathbb{R}_\mathsf{F} = \mathbb{R}_\lambda \cup \{\square \rightsquigarrow *\}$
System Fω	$\mathbb{R}_{\mathsf{F}\omega} = \mathbb{R}_\mathsf{F} \cup \{\square \rightsquigarrow \square\}$

Family of λ-calculi

$$\mathbb{S} = \{*, \Box\} \text{ (types, kinds)}, \ \mathbb{A} = \{* : \Box\}$$

λ-calculus	$\mathbb{R}$
Simply Typed	$\mathbb{R}_\lambda = \{* \rightsquigarrow *\}$
System F	$\mathbb{R}_\mathsf{F} = \mathbb{R}_\lambda \cup \{\Box \rightsquigarrow *\}$
System Fω	$\mathbb{R}_{\mathsf{F}\omega} = \mathbb{R}_\mathsf{F} \cup \{\Box \rightsquigarrow \Box\}$
Calculus of Constructions (CC)	$\mathbb{R}_\mathsf{CC} = \mathbb{R}_{\mathsf{F}\omega} \cup \{* \rightsquigarrow \Box\}$,

$$\mathbb{R}_\lambda \subseteq \mathbb{R}_\mathsf{F} \subseteq \mathbb{R}_{\mathsf{F}\omega} \subseteq \mathbb{R}_\mathsf{CC}$$