Reynolds' Parametricity

Patricia Johann
Appalachian State University
`cs.appstate.edu/~johannp`

Based on joint work with Neil Ghani, Fredrik Nordvall Forsberg, Federico Orsanigo, and Tim Revell

OPLSS 2016

Course Outline

Topic: Reynolds' theory of parametric polymorphism for System F

Goals: - extract the fibrational essence of Reynolds' theory
- generalize Reynolds' construction to very general models

- **Lecture 1: Reynolds' theory of parametricity for System F**
- **Lecture 2: Introduction to fibrations**
- **Lecture 3: A bifibrational view of parametricity**
- **Lecture 4: Bifibrational parametric models for System F**

Course Outline

Topic: Reynolds' theory of parametric polymorphism for System F

Goals:
- **extract the fibrational essence of Reynolds' theory**
- **generalize Reynolds' construction to very general models**

- **Lecture 1: Reynolds' theory of parametricity for System F**
- **Lecture 2: Introduction to fibrations**
- **Lecture 3: A bifibrational view of parametricity**
- **Lecture 4: Bifibrational parametric models for System F**

Parametric Polymorphic Functions

- **Intuitively, parametric polymorphism captures uniform behavior of functions across all type instantiations**

Parametric Polymorphic Functions

- **Intuitively, parametric polymorphism captures uniform behavior of functions across all type instantiations**
- **Distinct from *ad hoc* polymorphism (e.g., Haskell type classes)**

Parametric Polymorphic Functions

- **Intuitively, parametric polymorphism captures uniform behavior of functions across all type instantiations**
- **Distinct from *ad hoc* polymorphism (e.g., Haskell type classes)**
- **Uniformity of parametric polymorphic functions means that they**
  - **must be given by a single algorithm that works across all types**

Parametric Polymorphic Functions

- **Intuitively, parametric polymorphism captures uniform behavior of functions across all type instantiations**
- **Distinct from *ad hoc* polymorphism (e.g., Haskell type classes)**
- **Uniformity of parametric polymorphic functions means that they**
  - **must be given by a single algorithm that works across all types**
  - **cannot make use of any type-specific operations (e.g., $+$, $\neg$)**

Parametric Polymorphic Functions

- **Intuitively, parametric polymorphism captures uniform behavior of functions across all type instantiations**
- **Distinct from *ad hoc* polymorphism (e.g., Haskell type classes)**
- **Uniformity of parametric polymorphic functions means that they**
  - **must be given by a single algorithm that works across all types**
  - **cannot make use of any type-specific operations (e.g., $+$, $\neg$)**
  - **must map related inputs to related outputs**

Syntax and Semantics

- **Syntax: The $\forall$ type constructor**

Syntax and Semantics

- Syntax: The $\forall$ type constructor
- Semantics: If $f : \forall\alpha.\tau$, then f maps related values to related values (and similarly for $f : \tau_1 \to \tau_2$)

Syntax and Semantics

- Syntax: The $\forall$ type constructor
- Semantics: If $f : \forall\alpha.\tau$, then f maps related values to related values (and similarly for $f : \tau_1 \rightarrow \tau_2$)
- This semantic "relatedness" requirement ensures that models of parametric polymorphism do not contain *ad hoc* functions

Syntax and Semantics

- **Syntax: The $\forall$ type constructor**
- **Semantics: If $f : \forall\alpha.\tau$, then f maps related values to related values (and similarly for $f : \tau_1 \to \tau_2$)**
- **This semantic "relatedness" requirement ensures that models of parametric polymorphism do not contain *ad hoc* functions**
- **That is, it ensures that $\forall$ really does mean a uniform "for all"!**

Reynolds' Programme

- **Paper: Types, abstraction, and parametric polymorphism (1983)**

Reynolds' Programme

- **Paper:** **Types, abstraction, and parametric polymorphism (1983)**
- **Construct:** **A set-theoretic model of parametric polymorphism for System F by giving, compositionally:**

Reynolds' Programme

- **Paper: Types, abstraction, and parametric polymorphism (1983)**
- **Construct: A set-theoretic model of parametric polymorphism for System F by giving, compositionally:**
  - **a set interpretation for every type**

Reynolds' Programme

- **Paper:** Types, abstraction, and parametric polymorphism (1983)
- **Construct:** A set-theoretic model of parametric polymorphism for System F by giving, compositionally:
  - a set interpretation for every type
  - a relational interpretation for every type

Reynolds' Programme

- **Paper:** **Types, abstraction, and parametric polymorphism (1983)**
- **Construct:** **A set-theoretic model of parametric polymorphism for System F by giving, compositionally:**
  - **a set interpretation for every type**
  - **a relational interpretation for every type**
  - **a set interpretation for every term**

Reynolds' Programme

- Paper: Types, abstraction, and parametric polymorphism (1983)
- Construct: A set-theoretic model of parametric polymorphism for System F by giving, compositionally:
  - a set interpretation for every type
  - a relational interpretation for every type
  - a set interpretation for every term
- Prove: An Abstraction Theorem

Reynolds' Programme

- **Paper:** Types, abstraction, and parametric polymorphism (1983)
- **Construct:** A set-theoretic model of parametric polymorphism for System F by giving, compositionally:
  - a set interpretation for every type
  - a relational interpretation for every type
  - a set interpretation for every term
- **Prove:** An Abstraction Theorem
  - Intuitively, if the arguments to a function are related at the relational interpretations of their types, then applying the function to them yields results that are related at the relational interpretation of the function's return type

A Caveat and Its Correction

- **However...**

A Caveat and Its Correction

- **However…**

  … Reynolds discovered that his own construction was flawed!

A Caveat and Its Correction

- However...

  ... Reynolds discovered that his own construction was flawed!

- Reynolds showed that no set-theoretic model of parametric polymorphism for System F can exist in classical set theory

A Caveat and Its Correction

- **However...**

  ... Reynolds discovered that his own construction was flawed!

- **Reynolds showed that no set-theoretic model of parametric polymorphism for System F can exist in classical set theory**

- **The next year, Andrew Pitts showed that such models do exist... provided we work in a constructive set theory**

A Caveat and Its Correction

- However...

  ... Reynolds discovered that his own construction was flawed!

- Reynolds showed that no set-theoretic model of parametric polymorphism for System F can exist in classical set theory

- The next year, Andrew Pitts showed that such models do exist... provided we work in a constructive set theory

- Subsequently, a number of models of parametric polymorphism for System F have been constructed

A Caveat and Its Correction

- **However...**

  ... Reynolds discovered that his own construction was flawed!

- **Reynolds showed that no set-theoretic model of parametric polymorphism for System F can exist in classical set theory**

- **The next year, Andrew Pitts showed that such models do exist... provided we work in a constructive set theory**

- **Subsequently, a number of models of parametric polymorphism for System F have been constructed**

- **We'll see one such model, based on bifibrations**

A Caveat and Its Correction

- However...

  ... Reynolds discovered that his own construction was flawed!

- Reynolds showed that no set-theoretic model of parametric polymorphism for System F can exist in classical set theory

- The next year, Andrew Pitts showed that such models do exist... provided we work in a constructive set theory

- Subsequently, a number of models of parametric polymorphism for System F have been constructed

- We'll see one such model, based on bifibrations

- This model inhabits a "sweet spot" between
  - having the simplicity of functorial models, and

A Caveat and Its Correction

- However...

  ... Reynolds discovered that his own construction was flawed!

- Reynolds showed that no set-theoretic model of parametric polymorphism for System F can exist in classical set theory

- The next year, Andrew Pitts showed that such models do exist... provided we work in a constructive set theory

- Subsequently, a number of models of parametric polymorphism for System F have been constructed

- We'll see one such model, based on bifibrations

- This model inhabits a "sweet spot" between
  - having the simplicity of functorial models, and
  - having enough structure to derive consequences of parametricity that serve as gold standard properties for "good" models

Type Contexts and Judgements

- A **type context** Δ is a list of type variables $\alpha_1, ..., \alpha_n$

Type Contexts and Judgements

- A **type context** Δ is a list of type variables $\alpha_1, \ldots, \alpha_n$
- A **type judgement** $\Delta \vdash \tau$ has
  - Δ a type context
  - τ a type

Type Contexts and Judgements

- A type context Δ is a list of type variables $\alpha_1, \ldots, \alpha_n$
- A type judgement $\Delta \vdash \tau$ has
  - Δ a type context
  - τ a type
- Type judgements are defined inductively:

$$\frac{\alpha_i \in \Delta}{\Delta \vdash \alpha_i} \qquad \frac{\Delta \vdash \tau_1 \quad \Delta \vdash \tau_2}{\Delta \vdash \tau_1 \to \tau_2} \qquad \frac{\Delta, \alpha \vdash \tau}{\Delta \vdash \forall \alpha . \tau}$$

Type Contexts and Judgements

- A type context Δ is a list of type variables $\alpha_1, ..., \alpha_n$
- A type judgement $\Delta \vdash \tau$ has
  - Δ a type context
  - τ a type
- Type judgements are defined inductively:

$$\frac{\alpha_i \in \Delta}{\Delta \vdash \alpha_i} \qquad \frac{\Delta \vdash \tau_1 \quad \Delta \vdash \tau_2}{\Delta \vdash \tau_1 \to \tau_2} \qquad \frac{\Delta, \alpha \vdash \tau}{\Delta \vdash \forall \alpha . \tau}$$

- We consider α-convertible types equivalent

Term Contexts and Judgements - Part I

- A term context $\Delta \vdash \Gamma$ has
  - Δ a type context
  - $x_1, ..., x_m$ term variables
  - Γ of the form $x_1 : \tau_1, ..., x_m : \tau_m$
  - $\Delta \vdash \tau_i$ for each $i \in \{1, ..., m\}$

Term Contexts and Judgements - Part I

- A term context $\Delta \vdash \Gamma$ has
  - Δ a type context
  - $x_1, ..., x_m$ term variables
  - Γ of the form $x_1 : \tau_1, ..., x_m : \tau_m$
  - $\Delta \vdash \tau_i$ for each $i \in \{1, ..., m\}$
- A term judgement $\Delta; \Gamma \vdash t : \tau$ has
  - Δ a type context
  - $\Delta \vdash \Gamma$ a term context
  - $\Delta \vdash \tau$ a type judgement
  - t a term

Term Contexts and Judgements - Part II

- **Term judgements are defined inductively:**

$$\frac{\Delta \vdash \tau_i \quad x_i : \tau_i \in \Gamma}{\Delta; \Gamma \vdash x_i : \tau_i} \qquad \frac{\Delta; \Gamma, x : \tau_1 \vdash t : \tau_2}{\Delta; \Gamma \vdash \lambda x.t : \tau_1 \to \tau_2} \qquad \frac{\Delta; \Gamma \vdash t_1 : \tau_1 \quad \Delta; \Gamma \vdash t_2 : \tau_1 \to \tau_2}{\Delta; \Gamma \vdash t_2\, t_1 : \tau_2}$$

$$\frac{\Delta, \alpha; \Gamma \vdash t : \tau}{\Delta; \Gamma \vdash \Lambda\alpha.t : \forall\alpha.\tau} \qquad \frac{\Delta; \Gamma \vdash t : \forall\alpha.\tau_2 \quad \Delta \vdash \tau_1}{\Delta; \Gamma \vdash t\, \tau_1 : \tau_2[\alpha \mapsto \tau_1]}$$

Term Contexts and Judgements - Part II

- **Term judgements are defined inductively:**

$$\frac{\Delta \vdash \tau_i \quad x_i : \tau_i \in \Gamma}{\Delta; \Gamma \vdash x_i : \tau_i} \qquad \frac{\Delta; \Gamma, x : \tau_1 \vdash t : \tau_2}{\Delta; \Gamma \vdash \lambda x.t : \tau_1 \to \tau_2} \qquad \frac{\Delta; \Gamma \vdash t_1 : \tau_1 \quad \Delta; \Gamma \vdash t_2 : \tau_1 \to \tau_2}{\Delta; \Gamma \vdash t_2\, t_1 : \tau_2}$$

$$\frac{\Delta, \alpha; \Gamma \vdash t : \tau}{\Delta; \Gamma \vdash \Lambda\alpha.t : \forall\alpha.\tau} \qquad \frac{\Delta; \Gamma \vdash t : \forall\alpha.\tau_2 \quad \Delta \vdash \tau_1}{\Delta; \Gamma \vdash t\, \tau_1 : \tau_2[\alpha \mapsto \tau_1]}$$

- **Type abstraction requires that α does not appear (free) in Γ**

Term Contexts and Judgements - Part II

- **Term judgements are defined inductively:**

$$\frac{\Delta \vdash \tau_i \quad x_i : \tau_i \in \Gamma}{\Delta; \Gamma \vdash x_i : \tau_i} \qquad \frac{\Delta; \Gamma, x : \tau_1 \vdash t : \tau_2}{\Delta; \Gamma \vdash \lambda x.t : \tau_1 \to \tau_2} \qquad \frac{\Delta; \Gamma \vdash t_1 : \tau_1 \quad \Delta; \Gamma \vdash t_2 : \tau_1 \to \tau_2}{\Delta; \Gamma \vdash t_2\, t_1 : \tau_2}$$

$$\frac{\Delta, \alpha; \Gamma \vdash t : \tau}{\Delta; \Gamma \vdash \Lambda\alpha.t : \forall\alpha.\tau} \qquad \frac{\Delta; \Gamma \vdash t : \forall\alpha.\tau_2 \quad \Delta \vdash \tau_1}{\Delta; \Gamma \vdash t\, \tau_1 : \tau_2[\alpha \mapsto \tau_1]}$$

- **Type abstraction requires that α does not appear (free) in Γ**
- **$\tau_2[\alpha \mapsto \tau_1]$, $t[\alpha \mapsto \tau_1]$, and $t[x \mapsto y]$ denote (capture-free) substitution**

Conversion Rules - Part I

$$\frac{}{\Delta;\Gamma \vdash \lambda x.\, t = \lambda y.\, t[x \mapsto y] : \tau_1 \to \tau_2}\ (\alpha_\lambda) \qquad \frac{}{\Delta;\Gamma \vdash \Lambda\alpha_1.\, t = \Lambda\alpha_2.\, t[\alpha_1 \mapsto \alpha_2] : \forall\alpha_1.\tau}\ (\alpha_\Lambda)$$

$$\frac{}{\Delta;\Gamma \vdash (\lambda x.\, t)\ s = t[x \mapsto s] : \tau_2}\ (\beta_\lambda) \qquad \frac{}{\Delta;\Gamma \vdash (\Lambda\alpha.\, t)\tau_1 = t : \tau_2[\alpha \mapsto \tau_1]}\ (\beta_\Lambda)$$

$$\frac{x \notin FV(t)}{\Delta;\Gamma \vdash t = \lambda x.\, t\, x : \tau_1 \to \tau_2}\ (\eta_\lambda) \qquad \frac{\alpha \notin FTV(t)}{\Delta;\Gamma \vdash t = \Lambda\alpha.\, t\, \alpha : \forall\alpha.\tau}\ (\eta_\Lambda)$$

$$\frac{\Delta;\Gamma, x : \tau_1 \vdash t_1 = t_2 : \tau_2}{\Delta;\Gamma \vdash \lambda x.\, t_1 = \lambda x.\, t_2 : \tau_1 \to \tau_2}\ (\xi_\lambda) \qquad \frac{\Delta, \alpha;\Gamma \vdash t_1 = t_2 : \tau}{\Delta;\Gamma \vdash \Lambda\alpha.\, t_1 = \Lambda\alpha.\, t_2 : \forall\alpha.\tau}\ (\xi_\Lambda)$$

Conversion Rules - Part II

$$\frac{\Delta; \Gamma \vdash t_1 = t_2 : \tau_1 \to \tau_2 \qquad \Delta; \Gamma \vdash s_1 = s_2 : \tau_1}{\Delta; \Gamma \vdash t_1\, s_1 = t_2\, s_2 : \tau_2} \ (\text{cong}_\lambda)$$

$$\frac{\Delta; \Gamma \vdash t_1 = t_2 : \forall \alpha . \tau_2}{\Delta; \Gamma \vdash t_1\, \tau_1 = t_2\, \tau_1 : \tau_2[\alpha \mapsto \tau_1]} \ (\text{cong}_\Lambda)$$

$$\frac{}{\Delta; \Gamma \vdash t = t : \tau} \ (\text{refl}) \qquad \frac{\Delta; \Gamma \vdash s = t : \tau}{\Delta; \Gamma \vdash t = s : \tau} \ (\text{sym})$$

$$\frac{\Delta; \Gamma \vdash t = s : \tau \qquad \Gamma; \Delta \vdash s = u : \tau}{\Delta; \Gamma \vdash t = u : \tau} \ (\text{trans})$$

Reynolds' Semantics of Types - The Set Up

- **Reynolds defines two "parallel" semantics for System F types $\boldsymbol{\Delta} \vdash \boldsymbol{\tau}$**

Reynolds' Semantics of Types - The Set Up

- **Reynolds defines two "parallel" semantics for System F types $\Delta \vdash \tau$**
  - **an object semantics $[\![\Delta \vdash \tau]\!]_o : \mathsf{Set}^{|\Delta|} \to \mathsf{Set}$**

Reynolds' Semantics of Types - The Set Up

- Reynolds defines two "parallel" semantics for System F types $\boldsymbol{\Delta} \vdash \boldsymbol{\tau}$
  - an object semantics $[\![\boldsymbol{\Delta} \vdash \boldsymbol{\tau}]\!]_o : \mathsf{Set}^{|\boldsymbol{\Delta}|} \to \mathsf{Set}$
  - a relational semantics $[\![\boldsymbol{\Delta} \vdash \boldsymbol{\tau}]\!]_r : \mathsf{Rel}^{|\boldsymbol{\Delta}|} \to \mathsf{Rel}$

Reynolds' Semantics of Types - The Set Up

- Reynolds defines two "parallel" semantics for System F types $\Delta \vdash \tau$
  - an object semantics $[\![\Delta \vdash \tau]\!]_o : \mathsf{Set}^{|\Delta|} \to \mathsf{Set}$
  - a relational semantics $[\![\Delta \vdash \tau]\!]_r : \mathsf{Rel}^{|\Delta|} \to \mathsf{Rel}$
- Write
  - $S : \mathsf{Set}$ if S is a set
  - $R : \mathsf{Rel}$ if R is a relation
  - $R : \mathsf{Rel}(X, Y)$ if R is a relation on sets X and Y (i.e., $R \subseteq X \times Y$)

Reynolds' Semantics of Types - The Set Up

- Reynolds defines two "parallel" semantics for System F types $\Delta \vdash \tau$
  - an object semantics $[\![\Delta \vdash \tau]\!]_o : \mathsf{Set}^{|\Delta|} \to \mathsf{Set}$
  - a relational semantics $[\![\Delta \vdash \tau]\!]_r : \mathsf{Rel}^{|\Delta|} \to \mathsf{Rel}$
- Write
  - $S : \mathsf{Set}$ if S is a set
  - $R : \mathsf{Rel}$ if R is a relation
  - $R : \mathsf{Rel}(X, Y)$ if R is a relation on sets X and Y (i.e., $R \subseteq X \times Y$)
- Let
  - $\overline{X}$ be a $|\Delta|$-tuple of sets
  - $\overline{R}$ be a $|\Delta|$-tuple of relations
  - $R_i : \mathsf{Rel}(X_i, Y_i)$ for $i = 1, ..., |\Delta|$
  - $\mathsf{Eq}\, X = \{(x, x) \mid x \in X\}$

Reynolds' Semantics of Types

- **Type variables:** $[\![\boldsymbol{\Delta} \vdash \boldsymbol{\alpha_i}]\!]_o \overline{\boldsymbol{X}} \;=\; \boldsymbol{X_i}$ **and** $[\![\boldsymbol{\Delta} \vdash \boldsymbol{\alpha_i}]\!]_r \overline{\boldsymbol{R}} \;=\; \boldsymbol{R_i}$

Reynolds' Semantics of Types

- **Type variables:** $[\![\Delta \vdash \alpha_i]\!]_o \overline{X} \; = \; X_i$ and $[\![\Delta \vdash \alpha_i]\!]_r \overline{R} \; = \; R_i$
- **Arrow types:**
  - $[\![\Delta \vdash \tau_1 \to \tau_2]\!]_o \overline{X} = [\![\Delta \vdash \tau_1]\!]_o \overline{X} \to [\![\Delta \vdash \tau_2]\!]_o \overline{X}$

Reynolds' Semantics of Types

- **Type variables:** $[\![\Delta \vdash \alpha_i]\!]_o \overline{X} \;=\; X_i$ and $[\![\Delta \vdash \alpha_i]\!]_r \overline{R} \;=\; R_i$
- **Arrow types:**
  - $[\![\Delta \vdash \tau_1 \to \tau_2]\!]_o \overline{X} = [\![\Delta \vdash \tau_1]\!]_o \overline{X} \to [\![\Delta \vdash \tau_2]\!]_o \overline{X}$
  - $[\![\Delta \vdash \tau_1 \to \tau_2]\!]_r \overline{R} = \{(f, g) \mid (a, b) \in [\![\Delta \vdash \tau_1]\!]_r \overline{R} \Rightarrow (f\, a, g\, b) \in [\![\Delta \vdash \tau_2]\!]_r \overline{R}\}$

Reynolds' Semantics of Types

- **Type variables:** $[\![\Delta \vdash \alpha_i]\!]_o \overline{X} \;=\; X_i$ and $[\![\Delta \vdash \alpha_i]\!]_r \overline{R} \;=\; R_i$
- **Arrow types:**
  - $[\![\Delta \vdash \tau_1 \to \tau_2]\!]_o \overline{X} = [\![\Delta \vdash \tau_1]\!]_o \overline{X} \to [\![\Delta \vdash \tau_2]\!]_o \overline{X}$
  - $[\![\Delta \vdash \tau_1 \to \tau_2]\!]_r \overline{R} = \{(f, g) \mid (a, b) \in [\![\Delta \vdash \tau_1]\!]_r \overline{R} \Rightarrow (f\,a, g\,b) \in [\![\Delta \vdash \tau_2]\!]_r \overline{R}\}$

  Here, $f \in [\![\Delta \vdash \tau_1 \to \tau_2]\!]_o \overline{X}$ and $g \in [\![\Delta \vdash \tau_1 \to \tau_2]\!]_o \overline{Y}$

Reynolds' Semantics of Types

- Type variables: $[\![\Delta \vdash \alpha_i]\!]_o \overline{X} \;=\; X_i$ and $[\![\Delta \vdash \alpha_i]\!]_r \overline{R} \;=\; R_i$
- Arrow types:
  - $[\![\Delta \vdash \tau_1 \to \tau_2]\!]_o \overline{X} = [\![\Delta \vdash \tau_1]\!]_o \overline{X} \to [\![\Delta \vdash \tau_2]\!]_o \overline{X}$
  - $[\![\Delta \vdash \tau_1 \to \tau_2]\!]_r \overline{R} = \{(f, g) \mid (a, b) \in [\![\Delta \vdash \tau_1]\!]_r \overline{R} \Rightarrow (f\,a, g\,b) \in [\![\Delta \vdash \tau_2]\!]_r \overline{R}\}$

  Here, $f \in [\![\Delta \vdash \tau_1 \to \tau_2]\!]_o \overline{X}$ and $g \in [\![\Delta \vdash \tau_1 \to \tau_2]\!]_o \overline{Y}$
- Forall types:
  - $$[\![\Delta \vdash \forall\alpha.\tau]\!]_o \overline{X} = \{f : \prod_{S:\mathsf{Set}} [\![\Delta, \alpha \vdash \tau]\!]_o(\overline{X}, S) \mid \forall R' : \mathsf{Rel}(X', Y')\,.(f\,X', f\,Y') \in [\![\Delta, \alpha \vdash \tau]\!]_r(\overline{\mathsf{Eq}\ X}, R')\}$$

Reynolds' Semantics of Types

- **Type variables:** $[\![\Delta \vdash \alpha_i]\!]_o \overline{X} \;=\; X_i$ and $[\![\Delta \vdash \alpha_i]\!]_r \overline{R} \;=\; R_i$
- **Arrow types:**
  - $[\![\Delta \vdash \tau_1 \to \tau_2]\!]_o \overline{X} = [\![\Delta \vdash \tau_1]\!]_o \overline{X} \to [\![\Delta \vdash \tau_2]\!]_o \overline{X}$
  - $[\![\Delta \vdash \tau_1 \to \tau_2]\!]_r \overline{R} = \{(f,g) \mid (a,b) \in [\![\Delta \vdash \tau_1]\!]_r \overline{R} \Rightarrow (f\,a, g\,b) \in [\![\Delta \vdash \tau_2]\!]_r \overline{R}\}$

  Here, $f \in [\![\Delta \vdash \tau_1 \to \tau_2]\!]_o \overline{X}$ and $g \in [\![\Delta \vdash \tau_1 \to \tau_2]\!]_o \overline{Y}$
- **Forall types:**
  - $[\![\Delta \vdash \forall\alpha.\tau]\!]_o \overline{X} = \{f : \prod_{S:\mathsf{Set}} [\![\Delta, \alpha \vdash \tau]\!]_o (\overline{X}, S) \mid \forall R' : \mathsf{Rel}(X', Y')\,.(f X', f Y') \in [\![\Delta, \alpha \vdash \tau]\!]_r (\overline{\mathsf{Eq}\ X}, R')\}$
  - $[\![\Delta \vdash \forall\alpha.\tau]\!]_r \overline{R} = \{(f,g) \mid \forall R' : \mathsf{Rel}(X', Y')\,.\,(f X', g Y') \in [\![\Delta, \alpha \vdash \tau]\!]_r (\overline{R}, R')\}$

Reynolds' Semantics of Types

- **Type variables:** $[\![\Delta \vdash \alpha_i]\!]_o \overline{X} \; = \; X_i$ and $[\![\Delta \vdash \alpha_i]\!]_r \overline{R} \; = \; R_i$
- **Arrow types:**
  - $[\![\Delta \vdash \tau_1 \to \tau_2]\!]_o \overline{X} = [\![\Delta \vdash \tau_1]\!]_o \overline{X} \to [\![\Delta \vdash \tau_2]\!]_o \overline{X}$
  - $[\![\Delta \vdash \tau_1 \to \tau_2]\!]_r \overline{R} = \{(f, g) \mid (a, b) \in [\![\Delta \vdash \tau_1]\!]_r \overline{R} \Rightarrow (f\, a, g\, b) \in [\![\Delta \vdash \tau_2]\!]_r \overline{R}\}$

  Here, $f \in [\![\Delta \vdash \tau_1 \to \tau_2]\!]_o \overline{X}$ and $g \in [\![\Delta \vdash \tau_1 \to \tau_2]\!]_o \overline{Y}$
- **Forall types:**
  - $[\![\Delta \vdash \forall\alpha.\tau]\!]_o \overline{X} = \{f : \prod_{S:\mathsf{Set}} [\![\Delta, \alpha \vdash \tau]\!]_o(\overline{X}, S) \mid$
    $\forall R' : \mathsf{Rel}(X', Y')\,.(f X', f Y') \in [\![\Delta, \alpha \vdash \tau]\!]_r(\overline{\mathsf{Eq}\ X}, R')\}$
  - $[\![\Delta \vdash \forall\alpha.\tau]\!]_r \overline{R} = \{(f, g) \mid \forall R' : \mathsf{Rel}(X', Y')\,.\,(f X', g Y') \in [\![\Delta, \alpha \vdash \tau]\!]_r(\overline{R}, R')\}$

  Here, $f \in [\![\Delta \vdash \forall\alpha.\tau]\!]_o \overline{X}$ and $g \in [\![\Delta \vdash \forall\alpha.\tau]\!]_o \overline{Y}$

Some Observations

- **By construction, relational interpretations of functions (on types and on terms) map related inputs to related outputs**

Some Observations

- By construction, relational interpretations of functions (on types and on terms) map related inputs to related outputs
- If $\overline{R} : \mathsf{Rel}(\overline{X}, \overline{Y})$ then $[\![\Delta \vdash \tau]\!]_r \overline{R} : \mathsf{Rel}([\![\Delta \vdash \tau]\!]_o \overline{X}, [\![\Delta \vdash \tau]\!]_o \overline{Y})$

Some Observations

- **By construction, relational interpretations of functions (on types and on terms) map related inputs to related outputs**

- **If** $\overline{R} : \mathsf{Rel}(\overline{X}, \overline{Y})$ **then** $[\![\Delta \vdash \tau]\!]_r \overline{R} : \mathsf{Rel}([\![\Delta \vdash \tau]\!]_o \overline{X}, [\![\Delta \vdash \tau]\!]_o \overline{Y})$

- **The two interpretations of terms get progressively more intertwined:**

Some Observations

- By construction, relational interpretations of functions (on types and on terms) map related inputs to related outputs
- If $\overline{R} : \mathsf{Rel}(\overline{X}, \overline{Y})$ then $[\![\Delta \vdash \tau]\!]_r \overline{R} : \mathsf{Rel}([\![\Delta \vdash \tau]\!]_o \overline{X}, [\![\Delta \vdash \tau]\!]_o \overline{Y})$
- The two interpretations of terms get progressively more intertwined:
  - The object and relational interpretations of type variables are independent of one another

Some Observations

- By construction, relational interpretations of functions (on types and on terms) map related inputs to related outputs

- If $\overline{R} : \mathsf{Rel}(\overline{X}, \overline{Y})$ then $[\![\Delta \vdash \tau]\!]_r \overline{R} : \mathsf{Rel}([\![\Delta \vdash \tau]\!]_o \overline{X}, [\![\Delta \vdash \tau]\!]_o \overline{Y})$

- The two interpretations of terms get progressively more intertwined:
  - The object and relational interpretations of type variables are independent of one another
  - The object interpretation of an arrow type does not depend on its relational interpretation, but the relational interpretation of an arrow type *does* depend on its object interpretation

Some Observations

- By construction, relational interpretations of functions (on types and on terms) map related inputs to related outputs
- If $\overline{R} : \mathsf{Rel}(\overline{X}, \overline{Y})$ then $[\![\Delta \vdash \tau]\!]_r \overline{R} : \mathsf{Rel}([\![\Delta \vdash \tau]\!]_o \overline{X}, [\![\Delta \vdash \tau]\!]_o \overline{Y})$
- The two interpretations of terms get progressively more intertwined:
  - The object and relational interpretations of type variables are independent of one another
  - The object interpretation of an arrow type does not depend on its relational interpretation, but the relational interpretation of an arrow type *does* depend on its object interpretation
  - The object and relational interpretations of forall types depend crucially on one another

Some Observations

- By construction, relational interpretations of functions (on types and on terms) map related inputs to related outputs
- If $\overline{R} : \mathsf{Rel}(\overline{X}, \overline{Y})$ then $[\![\Delta \vdash \tau]\!]_r \overline{R} : \mathsf{Rel}([\![\Delta \vdash \tau]\!]_o \overline{X}, [\![\Delta \vdash \tau]\!]_o \overline{Y})$
- The two interpretations of terms get progressively more intertwined:
  - The object and relational interpretations of type variables are independent of one another
  - The object interpretation of an arrow type does not depend on its relational interpretation, but the relational interpretation of an arrow type *does* depend on its object interpretation
  - The object and relational interpretations of forall types depend crucially on one another
- So we do not really have two semantics, but rather a single interconnected semantics!

Identity Extension Lemma

- **Key for many applications of parametricity**

Identity Extension Lemma

- **Key for many applications of parametricity**
- **Intuitively, relational interpretations of types preserve equality**

Identity Extension Lemma

- Key for many applications of parametricity
- Intuitively, relational interpretations of types preserve equality
- Theorem (Identity Extension Lemma) For all $\Delta \vdash \tau$,

$$[\![\Delta \vdash \tau]\!]_r\,(\mathsf{Eq}\,X_1,\ldots,\mathsf{Eq}\,X_{|\Delta|}) = \mathsf{Eq}\,([\![\Delta \vdash \tau]\!]_o(X_1,\ldots,X_{|\Delta|}))$$

Reynolds' Semantics of Terms - The Set Up

- **Object and relational interpretations of term contexts**

$$\boldsymbol{\Gamma = x_1 : \tau_1, \ldots, x_m : \tau_m}$$

are given by

$$\boldsymbol{[\![\Delta \vdash \Gamma]\!]_o = [\![\Delta \vdash \tau_1]\!]_o \times \cdots \times [\![\Delta \vdash \tau_m]\!]_o}$$

and

$$\boldsymbol{[\![\Delta \vdash \Gamma]\!]_r = [\![\Delta \vdash \tau_1]\!]_r \times \cdots \times [\![\Delta \vdash \tau_m]\!]_r}$$

Reynolds' Semantics of Terms - The Set Up

- **Object and relational interpretations of term contexts**

$$\Gamma = x_1 : \tau_1, \ldots, x_m : \tau_m$$

are given by

$$[\![\Delta \vdash \Gamma]\!]_o = [\![\Delta \vdash \tau_1]\!]_o \times \cdots \times [\![\Delta \vdash \tau_m]\!]_o$$

and

$$[\![\Delta \vdash \Gamma]\!]_r = [\![\Delta \vdash \tau_1]\!]_r \times \cdots \times [\![\Delta \vdash \tau_m]\!]_r$$

- **An object interpretation of each term is a family of functions**

$$[\![\Delta; \Gamma \vdash t : \tau]\!]_o \overline{X} : [\![\Delta \vdash \Gamma]\!]_o \overline{X} \to [\![\Delta \vdash \tau]\!]_o \overline{X}$$

parameterized over a set environment $\overline{X}$

Reynolds' Semantics of Terms - The Set Up

- **Object and relational interpretations of term contexts**

$$\Gamma = x_1 : \tau_1, \ldots, x_m : \tau_m$$

are given by

$$[\![\Delta \vdash \Gamma]\!]_o = [\![\Delta \vdash \tau_1]\!]_o \times \cdots \times [\![\Delta \vdash \tau_m]\!]_o$$

and

$$[\![\Delta \vdash \Gamma]\!]_r = [\![\Delta \vdash \tau_1]\!]_r \times \cdots \times [\![\Delta \vdash \tau_m]\!]_r$$

- **An object interpretation of each term is a family of functions**

$$[\![\Delta; \Gamma \vdash t : \tau]\!]_o \overline{X} : [\![\Delta \vdash \Gamma]\!]_o \overline{X} \to [\![\Delta \vdash \tau]\!]_o \overline{X}$$

parameterized over a set environment $\overline{X}$

- **We'll sanity-check the definitions as we go along**

Reynolds' Semantics of Terms - variables

- If

$$\frac{}{\Delta; \Gamma \vdash x_i : \tau_i}$$

then

$$[\![\Delta; \Gamma \vdash x_i : \tau_i]\!]_o \, \overline{X} \, \overline{A} = A_i$$

Reynolds' Semantics of Terms - variables

- If

$$\overline{\boldsymbol{\Delta; \Gamma \vdash x_i : \tau_i}}$$

then

$$\llbracket \boldsymbol{\Delta; \Gamma \vdash x_i : \tau_i} \rrbracket_{\boldsymbol{o}} \, \overline{\boldsymbol{X}} \, \overline{\boldsymbol{A}} = \boldsymbol{A_i}$$

- This is sensible because we want

$$\llbracket \boldsymbol{\Delta; \Gamma \vdash x_i : \tau_i} \rrbracket_{\boldsymbol{o}} \, \overline{\boldsymbol{X}} \; : \; \llbracket \boldsymbol{\Delta \vdash \Gamma} \rrbracket_{\boldsymbol{o}} \, \overline{\boldsymbol{X}} \rightarrow \llbracket \boldsymbol{\Delta \vdash \tau_i} \rrbracket_{\boldsymbol{o}} \, \overline{\boldsymbol{X}}$$

Reynolds' Semantics of Terms - variables

- If

$$\overline{\Delta; \Gamma \vdash x_i : \tau_i}$$

  then

$$[\![\Delta; \Gamma \vdash x_i : \tau_i]\!]_o \, \overline{X} \, \overline{A} = A_i$$

- This is sensible because we want

$$[\![\Delta; \Gamma \vdash x_i : \tau_i]\!]_o \, \overline{X} \; : \; [\![\Delta \vdash \Gamma]\!]_o \, \overline{X} \to [\![\Delta \vdash \tau_i]\!]_o \, \overline{X}$$

  and because if $\overline{A} : [\![\Delta \vdash \Gamma]\!]_o \, \overline{X}$, then $A_i : [\![\Delta \vdash \tau_i]\!]_o \overline{X}$

Reynolds' Semantics of Terms - term abstractions

- **If**

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

  then

$$[\![\Delta; \Gamma \vdash \lambda x.t : \tau_1 \to \tau_2]\!]_o \, \overline{X} \, \overline{A} \, A = [\![\Delta; \Gamma, x : \tau_1 \vdash t : \tau_2]\!]_o \, \overline{X} \, (\overline{A}, A)$$

Reynolds' Semantics of Terms - term abstractions

- If

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

then

$$[\![\Delta;\Gamma \vdash \lambda x.t : \tau_1 \to \tau_2]\!]_o\, \overline{X}\, \overline{A}\, A = [\![\Delta;\Gamma, x:\tau_1 \vdash t:\tau_2]\!]_o\, \overline{X}\, (\overline{A}, A)$$

- This is sensible because we want

$$\begin{array}{rcl} [\![\Delta;\Gamma \vdash \lambda x.t : \tau_1 \to \tau_2]\!]_o\, \overline{X} & : & [\![\Delta \vdash \Gamma]\!]_o \overline{X} \to [\![\Delta \vdash \tau_1 \to \tau_2]\!]_o \overline{X} \\ & = & [\![\Delta \vdash \Gamma]\!]_o \overline{X} \to [\![\Delta \vdash \tau_1]\!]_o \overline{X} \to [\![\Delta \vdash \tau_2]\!]_o \overline{X} \end{array}$$

Reynolds' Semantics of Terms - term abstractions

- If

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

then

$$[\![\Delta; \Gamma \vdash \lambda x.t : \tau_1 \to \tau_2]\!]_o \, \overline{X} \, \overline{A} \, A = [\![\Delta; \Gamma, x : \tau_1 \vdash t : \tau_2]\!]_o \, \overline{X} \, (\overline{A}, A)$$

- This is sensible because we want

$$\begin{aligned} [\![\Delta; \Gamma \vdash \lambda x.t : \tau_1 \to \tau_2]\!]_o \overline{X} \quad &: \quad [\![\Delta \vdash \Gamma]\!]_o \overline{X} \to [\![\Delta \vdash \tau_1 \to \tau_2]\!]_o \overline{X} \\ &= \quad [\![\Delta \vdash \Gamma]\!]_o \overline{X} \to [\![\Delta \vdash \tau_1]\!]_o \overline{X} \to [\![\Delta \vdash \tau_2]\!]_o \overline{X} \end{aligned}$$

and because the IH gives

$$[\![\Delta; \Gamma, x : \tau_1 \vdash t : \tau_2]\!]_o \overline{X} : [\![\Delta \vdash \Gamma]\!]_o \overline{X} \times [\![\Delta \vdash \tau_1]\!]_o \overline{X} \to [\![\Delta \vdash \tau_2]\!]_o \overline{X}$$

Reynolds' Semantics of Terms - term applications

- **If**

$$\frac{\Delta; \Gamma \vdash t_1 : \tau_1 \quad \Delta; \Gamma \vdash t_2 : \tau_1 \to \tau_2}{\Delta; \Gamma \vdash t_2\, t_1 : \tau_2}$$

  then

  $$[\![\Delta; \Gamma \vdash t_2\, t_1 : \tau_2]\!]_o\, \overline{X}\, \overline{A} = [\![\Delta; \Gamma \vdash t_2 : \tau_1 \to \tau_2]\!]_o\, \overline{X}\, \overline{A}\, ([\![\Delta; \Gamma \vdash t_1 : \tau_1]\!]_o\, \overline{X}\, \overline{A})$$

Reynolds' Semantics of Terms - term applications

- If

$$\frac{\Delta; \Gamma \vdash t_1 : \tau_1 \quad \Delta; \Gamma \vdash t_2 : \tau_1 \to \tau_2}{\Delta; \Gamma \vdash t_2\, t_1 : \tau_2}$$

  then

  $[\![\Delta; \Gamma \vdash t_2\, t_1 : \tau_2]\!]_o\, \overline{X}\, \overline{A} = [\![\Delta; \Gamma \vdash t_2 : \tau_1 \to \tau_2]\!]_o\, \overline{X}\, \overline{A}\, ([\![\Delta; \Gamma \vdash t_1 : \tau_1]\!]_o\, \overline{X}\, \overline{A})$

- This is sensible because we want

$$[\![\Delta; \Gamma \vdash t_2\, t_1 : \tau_2]\!]_o\, \overline{X} : [\![\Delta \vdash \Gamma]\!]_o \overline{X} \to [\![\Delta \vdash \tau_2]\!]_o \overline{X}$$

Reynolds' Semantics of Terms - term applications

- If

$$\frac{\Delta;\Gamma \vdash t_1 : \tau_1 \quad \Delta;\Gamma \vdash t_2 : \tau_1 \to \tau_2}{\Delta;\Gamma \vdash t_2\, t_1 : \tau_2}$$

  then

  $$[\![\Delta;\Gamma \vdash t_2\, t_1 : \tau_2]\!]_o\, \overline{X}\, \overline{A} = [\![\Delta;\Gamma \vdash t_2 : \tau_1 \to \tau_2]\!]_o\, \overline{X}\, \overline{A}\, ([\![\Delta;\Gamma \vdash t_1 : \tau_1]\!]_o\, \overline{X}\, \overline{A})$$

- This is sensible because we want

$$[\![\Delta;\Gamma \vdash t_2\, t_1 : \tau_2]\!]_o\, \overline{X} : [\![\Delta \vdash \Gamma]\!]_o \overline{X} \to [\![\Delta \vdash \tau_2]\!]_o \overline{X}$$

  and because the IH gives

$$\begin{aligned} [\![\Delta;\Gamma \vdash t_2 : \tau_1 \to \tau_2]\!]_o\, \overline{X} \quad : \quad & [\![\Delta \vdash \Gamma]\!]_o\, \overline{X} \to [\![\Delta \vdash \tau_1 \to \tau_2]\!]_o\, \overline{X} \\ = \quad & [\![\Delta \vdash \Gamma]\!]_o \overline{X} \to [\![\Delta \vdash \tau_1]\!]_o\, \overline{X} \to [\![\Delta \vdash \tau_2]\!]_o\, \overline{X} \end{aligned}$$

Reynolds' Semantics of Terms - term applications

- If

$$\frac{\Delta; \Gamma \vdash t_1 : \tau_1 \quad \Delta; \Gamma \vdash t_2 : \tau_1 \to \tau_2}{\Delta; \Gamma \vdash t_2\, t_1 : \tau_2}$$

  then

  $[\![\Delta; \Gamma \vdash t_2\, t_1 : \tau_2]\!]_o\, \overline{X}\, \overline{A} = [\![\Delta; \Gamma \vdash t_2 : \tau_1 \to \tau_2]\!]_o\, \overline{X}\, \overline{A}\, ([\![\Delta; \Gamma \vdash t_1 : \tau_1]\!]_o\, \overline{X}\, \overline{A})$

- This is sensible because we want

$$[\![\Delta; \Gamma \vdash t_2\, t_1 : \tau_2]\!]_o\, \overline{X} : [\![\Delta \vdash \Gamma]\!]_o \overline{X} \to [\![\Delta \vdash \tau_2]\!]_o \overline{X}$$

  and because the IH gives

$$\begin{array}{rcl} [\![\Delta; \Gamma \vdash t_2 : \tau_1 \to \tau_2]\!]_o\, \overline{X} & : & [\![\Delta \vdash \Gamma]\!]_o\, \overline{X} \to [\![\Delta \vdash \tau_1 \to \tau_2]\!]_o\, \overline{X} \\ & = & [\![\Delta \vdash \Gamma]\!]_o \overline{X} \to [\![\Delta \vdash \tau_1]\!]_o\, \overline{X} \to [\![\Delta \vdash \tau_2]\!]_o\, \overline{X} \end{array}$$

  and

$$[\![\Delta; \Gamma \vdash t_1 : \tau_1]\!]_o\, \overline{X} : [\![\Delta \vdash \Gamma]\!]_o\, \overline{X} \to [\![\Delta \vdash \tau_1]\!]_o \overline{X}$$

Taking Stock

- **So far, term interpretations are all in the required sets**

Taking Stock

- **So far, term interpretations are all in the required sets**
- **But when Reynolds interpreted type abstractions and applications**

Taking Stock

- **So far, term interpretations are all in the required sets**
- **But when Reynolds interpreted type abstractions and applications**

... and tried to show that term interpretations are in the required sets

Taking Stock

- **So far, term interpretations are all in the required sets**
- **But when Reynolds interpreted type abstractions and applications**

... and tried to show that term interpretations are in the required sets

... he ran into problems

Reynolds' Semantics of Terms - type abstractions

- If

$$\frac{\Delta, \alpha; \Gamma \vdash t : \tau}{\Delta; \Gamma \vdash \Lambda\alpha.t : \forall\alpha.\tau}$$

then

$$[\![\Delta; \Gamma \vdash \Lambda\alpha.t : \forall\alpha.\tau]\!]_o\, \overline{X}\, \overline{A} = \Pi_{S:\mathsf{Set}}[\![\Delta, \alpha; \Gamma \vdash t : \tau]\!]_o\, (\overline{X}, S)\, \overline{A}$$

Reynolds' Semantics of Terms - type abstractions

- If

$$\frac{\Delta, \alpha; \Gamma \vdash t : \tau}{\Delta; \Gamma \vdash \Lambda\alpha.t : \forall\alpha.\tau}$$

then

$$[\![\Delta; \Gamma \vdash \Lambda\alpha.t : \forall\alpha.\tau]\!]_o \, \overline{X} \, \overline{A} = \Pi_{S:\mathsf{Set}}[\![\Delta, \alpha; \Gamma \vdash t : \tau]\!]_o \, (\overline{X}, S) \, \overline{A}$$

- This is sensible because we want

$$\begin{aligned} [\![\Delta; \Gamma \vdash \Lambda\alpha.t : \forall\alpha.\tau]\!]_o \, \overline{X} \quad &: \quad [\![\Delta \vdash \Gamma]\!]_o \, \overline{X} \to [\![\Delta \vdash \forall\alpha.\tau]\!]_o \, \overline{X} \\ &= \quad [\![\Delta \vdash \Gamma]\!]_o \, \overline{X} \to \{f : \Pi_{S:\mathsf{Set}}[\![\Delta, \alpha \vdash \tau]\!]_o(\overline{X}, S) \mid \ldots\} \end{aligned}$$

Reynolds' Semantics of Terms - type abstractions

- **If**

$$\frac{\Delta, \alpha; \Gamma \vdash t : \tau}{\Delta; \Gamma \vdash \Lambda\alpha.t : \forall\alpha.\tau}$$

then

$$[\![\Delta; \Gamma \vdash \Lambda\alpha.t : \forall\alpha.\tau]\!]_o\, \overline{X}\, \overline{A} = \Pi_{S:\mathsf{Set}}[\![\Delta, \alpha; \Gamma \vdash t : \tau]\!]_o\, (\overline{X}, S)\, \overline{A}$$

- **This is sensible because we want**

$$\begin{array}{rcl} [\![\Delta; \Gamma \vdash \Lambda\alpha.t : \forall\alpha.\tau]\!]_o\, \overline{X} & : & [\![\Delta \vdash \Gamma]\!]_o\, \overline{X} \to [\![\Delta \vdash \forall\alpha.\tau]\!]_o\, \overline{X} \\ & = & [\![\Delta \vdash \Gamma]\!]_o\, \overline{X} \to \{f : \Pi_{S:\mathsf{Set}}[\![\Delta, \alpha \vdash \tau]\!]_o(\overline{X}, S) \mid \ldots\} \end{array}$$

and because α not free in Γ implies

$$\begin{array}{rcl} [\![\Delta, \alpha; \Gamma \vdash t : \tau]\!]_o\, (\overline{X}, S) & : & [\![\Delta, \alpha \vdash \Gamma]\!]_o\, (\overline{X}, S) \to [\![\Delta, \alpha \vdash \tau]\!]_o\, (\overline{X}, S) \\ & = & [\![\Delta \vdash \Gamma]\!]_o\, \overline{X} \to [\![\Delta, \alpha \vdash \tau]\!]_o\, (\overline{X}, S) \end{array}$$

Reynolds' Semantics of Terms - type abstractions

- If

$$\frac{\Delta, \alpha; \Gamma \vdash t : \tau}{\Delta; \Gamma \vdash \Lambda\alpha.t : \forall\alpha.\tau}$$

  then

$$[\![\Delta; \Gamma \vdash \Lambda\alpha.t : \forall\alpha.\tau]\!]_o\, \overline{X}\, \overline{A} = \Pi_{S:\mathsf{Set}} [\![\Delta, \alpha; \Gamma \vdash t : \tau]\!]_o\, (\overline{X}, S)\, \overline{A}$$

- This is sensible because we want

$$\begin{array}{rcl} [\![\Delta; \Gamma \vdash \Lambda\alpha.t : \forall\alpha.\tau]\!]_o\, \overline{X} & : & [\![\Delta \vdash \Gamma]\!]_o\, \overline{X} \to [\![\Delta \vdash \forall\alpha.\tau]\!]_o\, \overline{X} \\ & = & [\![\Delta \vdash \Gamma]\!]_o\, \overline{X} \to \{f : \Pi_{S:\mathsf{Set}} [\![\Delta, \alpha \vdash \tau]\!]_o(\overline{X}, S) \mid \ldots\} \end{array}$$

  and because α not free in Γ implies

$$\begin{array}{rcl} [\![\Delta, \alpha; \Gamma \vdash t : \tau]\!]_o\, (\overline{X}, S) & : & [\![\Delta, \alpha \vdash \Gamma]\!]_o\, (\overline{X}, S) \to [\![\Delta, \alpha \vdash \tau]\!]_o\, (\overline{X}, S) \\ & = & [\![\Delta \vdash \Gamma]\!]_o\, \overline{X} \to [\![\Delta, \alpha \vdash \tau]\!]_o\, (\overline{X}, S) \end{array}$$

- But now we'd have to check that the condition after the vertical bar in the set interpretation of a $\forall$-type holds...

Reynolds' Semantics of Terms - type applications

- **If**

$$\frac{\Delta; \Gamma \vdash t : \forall \alpha . \tau_2 \quad \Delta \vdash \tau_1}{\Delta; \Gamma \vdash t\, \tau_1 : \tau_2[\alpha \mapsto \tau_1]}$$

  then

$$[\![\Delta; \Gamma \vdash t\, \tau_1 : \tau_2[\alpha \mapsto \tau_1]]\!]_o\, \overline{X}\, \overline{A} = [\![\Delta; \Gamma \vdash t : \forall \alpha . \tau_2]\!]_o\, \overline{X}\, \overline{A}\, ([\![\Delta \vdash \tau_1]\!]_o\, \overline{X})$$

Reynolds' Semantics of Terms - type applications

- If

$$\frac{\Delta; \Gamma \vdash t : \forall \alpha . \tau_2 \quad \Delta \vdash \tau_1}{\Delta; \Gamma \vdash t\, \tau_1 : \tau_2[\alpha \mapsto \tau_1]}$$

  then

$$[\![\Delta; \Gamma \vdash t\, \tau_1 : \tau_2[\alpha \mapsto \tau_1]]\!]_o\, \overline{X}\, \overline{A} = [\![\Delta; \Gamma \vdash t : \forall \alpha . \tau_2]\!]_o\, \overline{X}\, \overline{A}\, ([\![\Delta \vdash \tau_1]\!]_o\, \overline{X})$$

- This is sensible because we want

$$[\![\Delta; \Gamma \vdash t\, \tau_1 : \tau_2[\alpha \mapsto \tau_1]]\!]_o \overline{X} \;:\; [\![\Delta \vdash \Gamma]\!]_o \overline{X} \to [\![\Delta \vdash \tau_2[\alpha \mapsto \tau_1]]\!]_o \overline{X}$$

Reynolds' Semantics of Terms - type applications

- **If**

$$\frac{\Delta; \Gamma \vdash t : \forall \alpha . \tau_2 \qquad \Delta \vdash \tau_1}{\Delta; \Gamma \vdash t\, \tau_1 : \tau_2[\alpha \mapsto \tau_1]}$$

  then

$$[\![\Delta; \Gamma \vdash t\, \tau_1 : \tau_2[\alpha \mapsto \tau_1]]\!]_o\, \overline{X}\, \overline{A} = [\![\Delta; \Gamma \vdash t : \forall \alpha . \tau_2]\!]_o\, \overline{X}\, \overline{A}\, ([\![\Delta \vdash \tau_1]\!]_o\, \overline{X})$$

- **This is sensible because we want**

$$[\![\Delta; \Gamma \vdash t\, \tau_1 : \tau_2[\alpha \mapsto \tau_1]]\!]_o \overline{X} \; : \; [\![\Delta \vdash \Gamma]\!]_o \overline{X} \to [\![\Delta \vdash \tau_2[\alpha \mapsto \tau_1]]\!]_o \overline{X}$$

  and because

$$\begin{aligned} [\![\Delta; \Gamma \vdash t : \forall \alpha . \tau_2]\!]_o\, \overline{X} \quad &: \quad [\![\Delta \vdash \Gamma]\!]_o\, \overline{X} \to [\![\Delta \vdash \forall \alpha . \tau_2]\!]_o\, \overline{X} \\ &= \quad [\![\Delta \vdash \Gamma]\!]_o\, \overline{X} \to \{f : \Pi_{S:\mathsf{Set}} [\![\Delta, \alpha \vdash \tau_2]\!]_o (\overline{X}, S) | \ldots\} \end{aligned}$$

Reynolds' Semantics of Terms - type applications

- If

$$\frac{\Delta; \Gamma \vdash t : \forall \alpha. \tau_2 \quad \Delta \vdash \tau_1}{\Delta; \Gamma \vdash t\, \tau_1 : \tau_2[\alpha \mapsto \tau_1]}$$

  then

$$[\![\Delta; \Gamma \vdash t\, \tau_1 : \tau_2[\alpha \mapsto \tau_1]]\!]_o\, \overline{X}\, \overline{A} = [\![\Delta; \Gamma \vdash t : \forall \alpha. \tau_2]\!]_o\, \overline{X}\, \overline{A}\, ([\![\Delta \vdash \tau_1]\!]_o\, \overline{X})$$

- This is sensible because we want

$$[\![\Delta; \Gamma \vdash t\, \tau_1 : \tau_2[\alpha \mapsto \tau_1]]\!]_o \overline{X} \;:\; [\![\Delta \vdash \Gamma]\!]_o \overline{X} \to [\![\Delta \vdash \tau_2[\alpha \mapsto \tau_1]]\!]_o \overline{X}$$

  and because

$$\begin{aligned} [\![\Delta; \Gamma \vdash t : \forall \alpha. \tau_2]\!]_o\, \overline{X} \quad &: \quad [\![\Delta \vdash \Gamma]\!]_o\, \overline{X} \to [\![\Delta \vdash \forall \alpha. \tau_2]\!]_o\, \overline{X} \\ &= \quad [\![\Delta \vdash \Gamma]\!]_o\, \overline{X} \to \{f : \Pi_{S:\mathsf{Set}} [\![\Delta, \alpha \vdash \tau_2]\!]_o(\overline{X}, S) | \ldots\} \end{aligned}$$

- To type-check this, we'd need to show

$$[\![\Delta; \Gamma \vdash t : \forall \alpha. \tau_2]\!]_o\, \overline{X}\, \overline{A}\, ([\![\Delta \vdash \tau_1]\!]_o\, \overline{X}) : [\![\Delta \vdash \tau_2[\alpha \mapsto \tau_1]]\!]_o\, \overline{X}$$

Reynolds' Semantics of Terms - type applications

- If

$$\frac{\Delta;\Gamma \vdash t : \forall\alpha.\tau_2 \quad \Delta \vdash \tau_1}{\Delta;\Gamma \vdash t\,\tau_1 : \tau_2[\alpha \mapsto \tau_1]}$$

then

$$[\![\Delta;\Gamma \vdash t\,\tau_1 : \tau_2[\alpha \mapsto \tau_1]]\!]_o\,\overline{X}\,\overline{A} = [\![\Delta;\Gamma \vdash t : \forall\alpha.\tau_2]\!]_o\,\overline{X}\,\overline{A}\,([\![\Delta \vdash \tau_1]\!]_o\,\overline{X})$$

- This is sensible because we want

$$[\![\Delta;\Gamma \vdash t\,\tau_1 : \tau_2[\alpha \mapsto \tau_1]]\!]_o\overline{X} \;:\; [\![\Delta \vdash \Gamma]\!]_o\overline{X} \to [\![\Delta \vdash \tau_2[\alpha \mapsto \tau_1]]\!]_o\overline{X}$$

and because

$$\begin{aligned} [\![\Delta;\Gamma \vdash t : \forall\alpha.\tau_2]\!]_o\,\overline{X} \quad &: \quad [\![\Delta \vdash \Gamma]\!]_o\,\overline{X} \to [\![\Delta \vdash \forall\alpha.\tau_2]\!]_o\,\overline{X} \\ &= \quad [\![\Delta \vdash \Gamma]\!]_o\,\overline{X} \to \{f : \Pi_{S:\mathsf{Set}}[\![\Delta,\alpha \vdash \tau_2]\!]_o\,(\overline{X},S) | \ldots\} \end{aligned}$$

- To type-check this, we'd need to show

$$[\![\Delta;\Gamma \vdash t : \forall\alpha.\tau_2]\!]_o\,\overline{X}\,\overline{A}\,([\![\Delta \vdash \tau_1]\!]_o\,\overline{X}) : [\![\Delta \vdash \tau_2[\alpha \mapsto \tau_1]]\!]_o\,\overline{X}$$

- But this *assumes* the interpretation of type abstractions is sensible...

What About Type Abstractions and Applications?

- **Due to size considerations, Reynolds cannot interpret $\forall\alpha.\tau$ as a set of the form $\Pi_{S \in \mathsf{Set}} S$ for the usual set-theoretic product**

What About Type Abstractions and Applications?

- Due to size considerations, Reynolds cannot interpret $\forall\alpha.\tau$ as a set of the form $\Pi_{S\in\mathsf{Set}}S$ for the usual set-theoretic product
  - α would have to range over *all* sets interpreting types... including the set interpreting $\forall\alpha.\tau$!

What About Type Abstractions and Applications?

- **Due to size considerations, Reynolds cannot interpret $\forall\alpha.\tau$ as a set of the form $\Pi_{S\in\mathsf{Set}}S$ for the usual set-theoretic product**
  - **α would have to range over *all* sets interpreting types... including the set interpreting $\forall\alpha.\tau$!**
  - **This is impossible!**

What About Type Abstractions and Applications?

- **Due to size considerations, Reynolds cannot interpret $\forall\alpha.\tau$ as a set of the form $\Pi_{S\in\mathsf{Set}} S$ for the usual set-theoretic product**
  - **α would have to range over *all* sets interpreting types... including the set interpreting $\forall\alpha.\tau$!**
  - **This is impossible!**
- **Idea: Maybe a weaker notion of "large" product can interpret $\forall\alpha.\tau$ while still preserving the usual binary product and function space?**

What About Type Abstractions and Applications?

- **Due to size considerations, Reynolds cannot interpret $\forall\alpha.\tau$ as a set of the form $\Pi_{S\in\mathsf{Set}}S$ for the usual set-theoretic product**
  - **α would have to range over *all* sets interpreting types... including the set interpreting $\forall\alpha.\tau$!**
  - **This is impossible!**
- **Idea: Maybe a weaker notion of "large" product can interpret $\forall\alpha.\tau$ while still preserving the usual binary product and function space?**
- **In order to exclude *ad hoc* polymorphic functions from his model, Reynolds restricts it by imposing a so-called parametricity property**

What About Type Abstractions and Applications?

- **Due to size considerations, Reynolds cannot interpret $\forall\alpha.\tau$ as a set of the form $\Pi_{S\in\mathsf{Set}}S$ for the usual set-theoretic product**
  - **α would have to range over *all* sets interpreting types... including the set interpreting $\forall\alpha.\tau$!**
  - **This is impossible!**
- **Idea: Maybe a weaker notion of "large" product can interpret $\forall\alpha.\tau$ while still preserving the usual binary product and function space?**
- **In order to exclude *ad hoc* polymorphic functions from his model, Reynolds restricts it by imposing a so-called parametricity property**
- **This leads to the interpretations we have seen**

What About Type Abstractions and Applications?

- Due to size considerations, Reynolds cannot interpret $\forall\alpha.\tau$ as a set of the form $\Pi_{S\in\mathsf{Set}}S$ for the usual set-theoretic product
  - α would have to range over *all* sets interpreting types... including the set interpreting $\forall\alpha.\tau$!
  - This is impossible!
- Idea: Maybe a weaker notion of "large" product can interpret $\forall\alpha.\tau$ while still preserving the usual binary product and function space?
- In order to exclude *ad hoc* polymorphic functions from his model, Reynolds restricts it by imposing a so-called parametricity property
- This leads to the interpretations we have seen
- Conjecturing that these definitions give a sensible model, Reynolds proves his Abstraction Theorem

Problems in Parametricity Paradise

- **The next year Reynolds discovered that there can be no set model of System F in which**
  - **$\times$ is interpreted as the usual binary product**
  - **$\rightarrow$ is the interpreted as the usual function space**
  - **$\forall\alpha.\tau$ is interpreted as a possibly restricted "large" product**

Problems in Parametricity Paradise

- **The next year Reynolds discovered that there can be no set model of System F in which**
  - **$\times$ is interpreted as the usual binary product**
  - **$\rightarrow$ is the interpreted as the usual function space**
  - **$\forall \alpha . \tau$ is interpreted as a possibly restricted "large" product**
- **This is the case no matter what notion of "parametric" is used to restrict "large" products to exclude *ad hoc* functions!**

Problems in Parametricity Paradise

- The next year Reynolds discovered that there can be no set model of System F in which
  - $\times$ is interpreted as the usual binary product
  - $\rightarrow$ is the interpreted as the usual function space
  - $\forall \alpha . \tau$ is interpreted as a possibly restricted "large" product
- This is the case no matter what notion of "parametric" is used to restrict "large" products to exclude *ad hoc* functions!
- Reynolds proved this working in a classical set theory

Problems in Parametricity Paradise

- The next year Reynolds discovered that there can be no set model of System F in which
  - $\times$ is interpreted as the usual binary product
  - $\rightarrow$ is the interpreted as the usual function space
  - $\forall\alpha.\tau$ is interpreted as a possibly restricted "large" product
- This is the case no matter what notion of "parametric" is used to restrict "large" products to exclude *ad hoc* functions!
- Reynolds proved this working in a classical set theory
- In 1987, Andrew Pitts showed that set models of System F do exist in constructive set theories

Problems in Parametricity Paradise

- The next year Reynolds discovered that there can be no set model of System F in which
  - $\times$ is interpreted as the usual binary product
  - $\rightarrow$ is the interpreted as the usual function space
  - $\forall\alpha.\tau$ is interpreted as a possibly restricted "large" product
- This is the case no matter what notion of "parametric" is used to restrict "large" products to exclude *ad hoc* functions!
- Reynolds proved this working in a classical set theory
- In 1987, Andrew Pitts showed that set models of System F do exist in constructive set theories
- We won't look at constructive set models of System F in this course

Problems in Parametricity Paradise

- **The next year Reynolds discovered that there can be no set model of System F in which**
  - **$\times$ is interpreted as the usual binary product**
  - **$\to$ is the interpreted as the usual function space**
  - **$\forall\alpha.\tau$ is interpreted as a possibly restricted "large" product**
- **This is the case no matter what notion of "parametric" is used to restrict "large" products to exclude *ad hoc* functions!**
- **Reynolds proved this working in a classical set theory**
- **In 1987, Andrew Pitts showed that set models of System F do exist in constructive set theories**
- **We won't look at constructive set models of System F in this course**
- **Instead, we'll just draw inspiration from Reynolds' ideas**

The Abstraction Theorem

- **Formalizes uniformity of parametric polymorphism**

The Abstraction Theorem

- **Formalizes uniformity of parametric polymorphism**
- **Intuitively, every (interpretation of every) term is related to itself by the relational interpretation of its type**

The Abstraction Theorem

- **Formalizes uniformity of parametric polymorphism**
- **Intuitively, every (interpretation of every) term is related to itself by the relational interpretation of its type**
- **Theorem (Abstraction Theorem) Let $\overline{X}, \overline{Y} : \mathsf{Set}^{|\Delta|}$, $\overline{R} : \mathsf{Rel}^{|\Delta|}(\overline{X}, \overline{Y})$, $\overline{A} \in [\![\Delta \vdash \Gamma]\!]_o \overline{X}$, and $\overline{B} \in [\![\Delta \vdash \Gamma]\!]_o \overline{Y}$. For all $\Delta; \Gamma \vdash t : \tau$, if**

$$(\overline{A}, \overline{B}) \in [\![\Delta \vdash \Gamma]\!]_r \overline{R}$$

then

$$([\![\Delta; \Gamma \vdash t : \tau]\!]_o\ \overline{X}\ \overline{A}, [\![\Delta; \Gamma \vdash t : \tau]\!]_o\ \overline{Y}\ \overline{B})\ \in\ [\![\Delta \vdash \tau]\!]_r \overline{R}$$

The Abstraction Theorem

- Formalizes uniformity of parametric polymorphism
- Intuitively, every (interpretation of every) term is related to itself by the relational interpretation of its type
- Theorem (Abstraction Theorem) Let $\overline{X}, \overline{Y} : \mathsf{Set}^{|\Delta|}$, $\overline{R} : \mathsf{Rel}^{|\Delta|}(\overline{X}, \overline{Y})$, $\overline{A} \in [\![\Delta \vdash \Gamma]\!]_o \overline{X}$, and $\overline{B} \in [\![\Delta \vdash \Gamma]\!]_o \overline{Y}$. For all $\Delta; \Gamma \vdash t : \tau$,

  if
  $$(\overline{A}, \overline{B}) \in [\![\Delta \vdash \Gamma]\!]_r \overline{R}$$
  then
  $$([\![\Delta; \Gamma \vdash t : \tau]\!]_o\ \overline{X}\ \overline{A}, [\![\Delta; \Gamma \vdash t : \tau]\!]_o\ \overline{Y}\ \overline{B})\ \in\ [\![\Delta \vdash \tau]\!]_r \overline{R}$$
- This doesn't make complete sense because of missing interpretations...

The Abstraction Theorem

- Formalizes uniformity of parametric polymorphism
- Intuitively, every (interpretation of every) term is related to itself by the relational interpretation of its type
- Theorem (Abstraction Theorem) Let $\overline{X}, \overline{Y} : \mathsf{Set}^{|\Delta|}$, $\overline{R} : \mathsf{Rel}^{|\Delta|}(\overline{X}, \overline{Y})$, $\overline{A} \in [\![\Delta \vdash \Gamma]\!]_o \overline{X}$, and $\overline{B} \in [\![\Delta \vdash \Gamma]\!]_o \overline{Y}$. For all $\Delta; \Gamma \vdash t : \tau$, if

$$(\overline{A}, \overline{B}) \in [\![\Delta \vdash \Gamma]\!]_r \overline{R}$$

then

$$([\![\Delta; \Gamma \vdash t : \tau]\!]_o \, \overline{X} \, \overline{A}, [\![\Delta; \Gamma \vdash t : \tau]\!]_o \, \overline{Y} \, \overline{B}) \ \in \ [\![\Delta \vdash \tau]\!]_r \overline{R}$$

- This doesn't make complete sense because of missing interpretations...
- ... but a model of System F in which the Abstraction Theorem and Identity Extension Lemma hold is what Reynolds was aiming for

Coming Up

- **Introduction to (bi)fibrations**

Coming Up

- Introduction to (bi)fibrations
- View Reynolds' construction and results through the lens of the relations (bi)fibration on $\mathbf{Set}$

Coming Up

- **Introduction to (bi)fibrations**
- **View Reynolds' construction and results through the lens of the relations (bi)fibration on Set**
- **Generalize Reynolds' constructions to (bi)fibrational models of System F for which we can prove (fibrational versions of) the IEL and Abstraction Theorem**

Coming Up

- **Introduction to (bi)fibrations**
- **View Reynolds' construction and results through the lens of the relations (bi)fibration on $\mathbf{Set}$**
- **Generalize Reynolds' constructions to (bi)fibrational models of System F for which we can prove (fibrational versions of) the IEL and Abstraction Theorem**
- **Reynolds' construction is (ignoring size issues) such a model**

References

- Types, abstraction, and parametric polymorphism. J. Reynolds. Information Processing, 1983.
- Polymorphism is not set-theoretic. J. Reynolds. Semantics of Data Types, 1984.
- Polymorphism is set-theoretic, constructively. A. Pitts. CTCS'84.