System F

Alexandre Miquel — PPS & U. Paris 7

Alexandre.Miquel@pps.jussieu.fr

Types Summer School 2005

August 15–26 — Göteborg

Introduction

Introduction

- **System F:** independently discovered by

Introduction

- **System F:** independently discovered by

  Girard: System F (1970)

Introduction

- **System F:** independently discovered by

Girard:	System F	(1970)
Reynolds:	The polymorphic λ-calculus	(1974)

Introduction

- **System F:** independently discovered by

Girard:	System F	(1970)
Reynolds:	The polymorphic λ-calculus	(1974)

- Quite different motivations…

Girard:	Interpretation of second-order logic
Reynolds:	Functional programming

… connected by the Curry-Howard isomorphism

Introduction

- **System F:** independently discovered by

Girard:	System F	(1970)
Reynolds:	The polymorphic λ-calculus	(1974)

- Quite different motivations...

Girard:	Interpretation of second-order logic
Reynolds:	Functional programming

... connected by the Curry-Howard isomorphism

- Significant influence on the development of Type Theory

– Interpretation of higher-order logic	[Girard, Martin-Löf]
– Type:Type	[Martin-Löf 1971]
– Martin-Löf Type Theory	[1972, 1984, 1990, ...]
– The Calculus of Constructions	[Coquand 1984]

Part I

System F: Church-style presentation

System F syntax

Definition

Types $A, B \quad ::= \quad \alpha \mid A \to B \mid \forall\alpha\ B$

Terms $t, u \quad ::= \quad x$

$\mid \quad \lambda x : A\,.\,t \quad \mid \quad tu$ (term abstr./app.)

$\mid \quad \Lambda\alpha\,.\,t \quad \mid \quad tA$ (type abstr./app.)

System F syntax

Definition

Types	A, B	$::=$	α	$\mid$	$A \to B$	$\mid$	$\forall\alpha\ B$
Terms	t, u	$::=$	x				
		$\mid$	$\lambda x : A\,.\,t$	$\mid$	tu		(term abstr./app.)
		$\mid$	$\Lambda\alpha\,.\,t$	$\mid$	tA		(type abstr./app.)

Notations

- Set of free (term) variables: $FV(t)$
- Set of free type variables: $TV(t), \quad TV(A)$
- Term substitution: $u\{x := t\}$
- Type substitution: $u\{\alpha := A\}, \quad B\{\alpha := A\}$

Perform α-conversion to prevent captures of free (term/type) variables!

System F typing rules

Contexts $\Gamma \;::=\; x_1 : A_1,\ \ldots,\ x_n : A_n$

Typing judgments $\Gamma \vdash t : A$

System F typing rules

Contexts $\Gamma \quad ::= \quad x_1 : A_1, \; \ldots, \; x_n : A_n$

Typing judgments $\Gamma \vdash t : A$

$$\frac{}{\Gamma \vdash x : A} \; {\scriptstyle (x:A) \in \Gamma}$$

$$\frac{\Gamma, \; x : A \vdash t : B}{\Gamma \vdash \lambda x : A \, . \, t : A \to B} \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : B}$$

$$\frac{\Gamma \vdash t : B}{\Gamma \vdash \Lambda\alpha \, . \, t : \forall\alpha \; B} \; {\scriptstyle \alpha \notin TV(\Gamma)} \qquad \frac{\Gamma \vdash t : \forall\alpha \; B}{\Gamma \vdash tA : B\{\alpha := A\}}$$

System F typing rules

Contexts $\Gamma \quad ::= \quad x_1 : A_1, \quad \ldots, \quad x_n : A_n$

Typing judgments $\Gamma \vdash t : A$

$$\frac{}{\Gamma \vdash x : A} \; {\scriptstyle (x:A) \in \Gamma}$$

$$\frac{\Gamma, \; x : A \vdash t : B}{\Gamma \vdash \lambda x : A \, . \, t : A \to B} \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : B}$$

$$\frac{\Gamma \vdash t : B}{\Gamma \vdash \Lambda\alpha \, . \, t : \forall\alpha \; B} \; {\scriptstyle \alpha \notin TV(\Gamma)} \qquad \frac{\Gamma \vdash t : \forall\alpha \; B}{\Gamma \vdash tA : B\{\alpha := A\}}$$

- Declaration of type variables is implicit (for each $\alpha \in TV(\Gamma)$)
- Type variables could be declared explicitly: $\alpha : *$ (cf PTS)
- One rule for each syntactic construct $\Rightarrow$ System is syntax-directed

Example: the polymorphic identity

- Set: $\mathsf{id} \equiv \Lambda\alpha\,.\,\lambda x : \alpha\,.\,x$

Example: the polymorphic identity

- Set: $\mathsf{id} \equiv \Lambda\alpha\,.\,\lambda x:\alpha\,.\,x$

- One has:

$$\mathsf{id} \;:\; \forall\alpha\ (\alpha \to \alpha)$$

Example: the polymorphic identity

- Set: $\mathsf{id} \equiv \Lambda\alpha\,.\,\lambda x : \alpha\,.\,x$
- One has:

$$\begin{array}{lll} \mathsf{id} & : & \forall\alpha\ (\alpha \to \alpha) \\ \mathsf{id}\,B & : & B \to B \qquad \text{for any type } B \end{array}$$

Example: the polymorphic identity

- Set: $\text{id} \equiv \Lambda\alpha\,.\,\lambda x : \alpha\,.\,x$

- One has:

$$\begin{array}{lcll} \text{id} & : & \forall\alpha\ (\alpha \to \alpha) & \\ \text{id}\,B & : & B \to B & \text{for any type } B \\ \text{id}\,B\,u & : & B & \text{for any term } u : B \end{array}$$

Example: the polymorphic identity

- Set: $\text{id} \;\equiv\; \Lambda\alpha\,.\,\lambda x:\alpha\,.\,x$

- One has:

$$
\begin{array}{lcll}
\text{id} & : & \forall\alpha\;(\alpha\to\alpha) & \\
\text{id}\,B & : & B\to B & \text{for any type } B \\
\text{id}\,B\,u & : & B & \text{for any term } u : B
\end{array}
$$

- In particular, if we take $B \;\equiv\; \forall\alpha\;(\alpha\to\alpha)$ and $u \;\equiv\; \text{id}$

Example: the polymorphic identity

- Set: $\text{id} \equiv \Lambda\alpha\,.\,\lambda x : \alpha\,.\,x$
- One has:

$$\begin{array}{lcll} \text{id} & : & \forall\alpha\ (\alpha \to \alpha) & \\ \text{id}\,B & : & B \to B & \text{for any type } B \\ \text{id}\,B\,u & : & B & \text{for any term } u : B \end{array}$$

- In particular, if we take $B \equiv \forall\alpha\ (\alpha \to \alpha)$ and $u \equiv \text{id}$

$$\text{id}\,\big(\forall\alpha\ (\alpha \to \alpha)\big) \quad : \quad \forall\alpha\ (\alpha \to \alpha) \to \forall\alpha\ (\alpha \to \alpha)$$

Example: the polymorphic identity

- Set: $\quad \mathrm{id} \;\equiv\; \Lambda\alpha\,.\,\lambda x : \alpha\,.\,x$

- One has:

$$\begin{array}{lcll} \mathrm{id} & : & \forall\alpha\;(\alpha \to \alpha) & \\ \mathrm{id}\,B & : & B \to B & \text{for any type } B \\ \mathrm{id}\,B\,u & : & B & \text{for any term } u : B \end{array}$$

- In particular, if we take $\quad B \;\equiv\; \forall\alpha\;(\alpha \to \alpha) \quad$ and $\quad u \;\equiv\; \mathrm{id}$

$$\begin{array}{lcl} \mathrm{id}\,\big(\forall\alpha\;(\alpha \to \alpha)\big) & : & \forall\alpha\;(\alpha \to \alpha) \;\to\; \forall\alpha\;(\alpha \to \alpha) \\ \mathrm{id}\,\big(\forall\alpha\;(\alpha \to \alpha)\big)\,\mathrm{id} & : & \forall\alpha\;(\alpha \to \alpha) \end{array}$$

Example: the polymorphic identity

- Set: $\mathsf{id} \equiv \Lambda\alpha\,.\,\lambda x:\alpha\,.\,x$

- One has:

$$\begin{array}{lcll} \mathsf{id} & : & \forall\alpha\,(\alpha\to\alpha) & \\ \mathsf{id}\,B & : & B\to B & \text{for any type } B \\ \mathsf{id}\,B\,u & : & B & \text{for any term } u:B \end{array}$$

- In particular, if we take $B \equiv \forall\alpha\,(\alpha\to\alpha)$ and $u \equiv \mathsf{id}$

$$\begin{array}{lcl} \mathsf{id}\,(\forall\alpha\,(\alpha\to\alpha)) & : & \forall\alpha\,(\alpha\to\alpha) \to \forall\alpha\,(\alpha\to\alpha) \\ \mathsf{id}\,(\forall\alpha\,(\alpha\to\alpha))\,\mathsf{id} & : & \forall\alpha\,(\alpha\to\alpha) \end{array}$$

$\Rightarrow$ Type system is impredicative (or cyclic)

Properties

Properties

Substitutivity (for types/terms):

Properties

Substitutivity (for types/terms):

- $\Gamma \vdash u : B \quad \Rightarrow \quad \Gamma\{\alpha := A\} \vdash u\{\alpha := A\} : B\{\alpha := A\}$

Properties

Substitutivity (for types/terms):

- $\Gamma \vdash u : B \quad \Rightarrow \quad \Gamma\{\alpha := A\} \vdash u\{\alpha := A\} : B\{\alpha := A\}$
- $\Gamma, x : A \vdash u : B, \quad \Gamma \vdash t : A \quad \Rightarrow \quad \Gamma \vdash u\{x := t\} : B$

Properties

Substitutivity (for types/terms):

- $\Gamma \vdash u : B \quad \Rightarrow \quad \Gamma\{\alpha := A\} \vdash u\{\alpha := A\} : B\{\alpha := A\}$
- $\Gamma, x : A \vdash u : B, \quad \Gamma \vdash t : A \quad \Rightarrow \quad \Gamma \vdash u\{x := t\} : B$

Uniqueness of type

Properties

Substitutivity (for types/terms):

- $\Gamma \vdash u : B \quad \Rightarrow \quad \Gamma\{\alpha := A\} \vdash u\{\alpha := A\} : B\{\alpha := A\}$
- $\Gamma, x : A \vdash u : B, \quad \Gamma \vdash t : A \quad \Rightarrow \quad \Gamma \vdash u\{x := t\} : B$

Uniqueness of type

$\Gamma \vdash t : A, \quad \Gamma \vdash t : A' \quad \Rightarrow \quad A = A' \quad (\alpha\text{-conv.})$

Properties

Substitutivity (for types/terms):

- $\Gamma \vdash u : B \quad \Rightarrow \quad \Gamma\{\alpha := A\} \vdash u\{\alpha := A\} : B\{\alpha := A\}$
- $\Gamma, x : A \vdash u : B, \quad \Gamma \vdash t : A \quad \Rightarrow \quad \Gamma \vdash u\{x := t\} : B$

Uniqueness of type

$\Gamma \vdash t : A, \quad \Gamma \vdash t : A' \quad \Rightarrow \quad A = A' \quad (\alpha\text{-conv.})$

Decidability of type checking / type inference

Properties

Substitutivity (for types/terms):

- $\Gamma \vdash u : B \quad \Rightarrow \quad \Gamma\{\alpha := A\} \vdash u\{\alpha := A\} : B\{\alpha := A\}$
- $\Gamma, x : A \vdash u : B, \quad \Gamma \vdash t : A \quad \Rightarrow \quad \Gamma \vdash u\{x := t\} : B$

Uniqueness of type

$\Gamma \vdash t : A, \quad \Gamma \vdash t : A' \quad \Rightarrow \quad A = A' \quad (\alpha\text{-conv.})$

Decidability of type checking / type inference

1. Given Γ, t and A, decide whether $\Gamma \vdash t : A$ is derivable

Properties

Substitutivity (for types/terms):

- $\Gamma \vdash u : B \quad \Rightarrow \quad \Gamma\{\alpha := A\} \vdash u\{\alpha := A\} : B\{\alpha := A\}$
- $\Gamma, x : A \vdash u : B, \quad \Gamma \vdash t : A \quad \Rightarrow \quad \Gamma \vdash u\{x := t\} : B$

Uniqueness of type

$\Gamma \vdash t : A, \quad \Gamma \vdash t : A' \quad \Rightarrow \quad A = A' \quad (\alpha\text{-conv.})$

Decidability of type checking / type inference

1. Given Γ, t and A, decide whether $\Gamma \vdash t : A$ is derivable
2. Given Γ and t, compute a type A such that $\Gamma \vdash t : A$ if such a type exists, or fail otherwise.

Properties

Substitutivity (for types/terms):

- $\Gamma \vdash u : B \quad \Rightarrow \quad \Gamma\{\alpha := A\} \vdash u\{\alpha := A\} : B\{\alpha := A\}$
- $\Gamma, x : A \vdash u : B, \quad \Gamma \vdash t : A \quad \Rightarrow \quad \Gamma \vdash u\{x := t\} : B$

Uniqueness of type

$\Gamma \vdash t : A, \quad \Gamma \vdash t : A' \quad \Rightarrow \quad A = A' \quad (\alpha\text{-conv.})$

Decidability of type checking / type inference

1. Given Γ, t and A, decide whether $\Gamma \vdash t : A$ is derivable
2. Given Γ and t, compute a type A such that $\Gamma \vdash t : A$ if such a type exists, or fail otherwise.

Both problems are decidable

Reduction rules

Two kinds of redexes:

Reduction rules

Two kinds of redexes:

$$(\lambda x : A \,.\, t)u \quad \succ \quad t\{x := u\} \qquad \text{1st kind redex}$$

Reduction rules

Two kinds of redexes:

$$
\begin{array}{rcll}
(\lambda x : A \,.\, t)u & \succ & t\{x := u\} & \text{1st kind redex} \\
(\Lambda\alpha \,.\, t)A & \succ & t\{\alpha := A\} & \text{2nd kind redex}
\end{array}
$$

Reduction rules

Two kinds of redexes:

$$\begin{array}{rcll} (\lambda x : A \,.\, t)u & \succ & t\{x := u\} & \text{1st kind redex} \\ (\Lambda\alpha \,.\, t)A & \succ & t\{\alpha := A\} & \text{2nd kind redex} \end{array}$$

Other combinations of abstraction and application are meaningless (and rejected by typing)

Reduction rules

Two kinds of redexes:

$$
\begin{array}{rcll}
(\lambda x : A \,.\, t) u & \succ & t\{x := u\} & \text{1st kind redex} \\
(\Lambda \alpha \,.\, t) A & \succ & t\{\alpha := A\} & \text{2nd kind redex}
\end{array}
$$

Other combinations of abstraction and application are meaningless (and rejected by typing)

Definitions

Reduction rules

Two kinds of redexes:

$$(\lambda x : A\,.\,t)u \;\succ\; t\{x := u\} \qquad \text{1st kind redex}$$

$$(\Lambda\alpha\,.\,t)A \;\succ\; t\{\alpha := A\} \qquad \text{2nd kind redex}$$

Other combinations of abstraction and application are meaningless (and rejected by typing)

Definitions

- One step β-reduction $t \succ t'$ $\equiv$
  contextual closure of both rules above

Reduction rules

Two kinds of redexes:

$$(\lambda x : A \,.\, t)u \quad \succ \quad t\{x := u\} \qquad \text{1st kind redex}$$

$$(\Lambda\alpha \,.\, t)A \quad \succ \quad t\{\alpha := A\} \qquad \text{2nd kind redex}$$

Other combinations of abstraction and application are meaningless (and rejected by typing)

Definitions

- One step β-reduction $t \succ t'$ $\equiv$
  contextual closure of both rules above
- β-reduction $t \succ^* t'$ $\equiv$
  reflexive-transitive closure of $\succ$

Reduction rules

Two kinds of redexes:

$$\begin{array}{rcll} (\lambda x:A\,.\,t)u & \succ & t\{x := u\} & \text{1st kind redex} \\ (\Lambda\alpha\,.\,t)A & \succ & t\{\alpha := A\} & \text{2nd kind redex} \end{array}$$

Other combinations of abstraction and application are meaningless (and rejected by typing)

Definitions

- One step β-reduction $t \succ t'$ $\equiv$ contextual closure of both rules above
- β-reduction $t \succ^* t'$ $\equiv$ reflexive-transitive closure of $\succ$
- β-convertibility $t \simeq t'$ $\equiv$ reflexive-symmetric-transitive closure of $\succ$

Examples

Examples

- **The polymorphic identity, again**

Examples

- **The polymorphic identity, again**

$$\mathsf{id}\ B\ u \quad \equiv \quad (\Lambda\alpha\,.\,\lambda x\!:\!\alpha\,.\,x)\ B\ u$$

Examples

- **The polymorphic identity, again**

$$\text{id}\ B\ u \quad \equiv \quad (\Lambda\alpha\,.\,\lambda x\!:\!\alpha\,.\,x)\ B\ u \quad \succ \quad (\lambda x\!:\!B\,.\,x)\ u$$

Examples

- **The polymorphic identity, again**

$$\mathrm{id}\ B\ u \quad \equiv \quad (\Lambda\alpha\,.\,\lambda x\!:\!\alpha\,.\,x)\ B\ u \quad \succ \quad (\lambda x\!:\!B\,.\,x)\ u \quad \succ \quad u$$

Examples

- **The polymorphic identity, again**

$$\text{id}\ B\ u \quad \equiv \quad (\Lambda\alpha\,.\,\lambda x : \alpha\,.\,x)\ B\ u \quad \succ \quad (\lambda x : B\,.\,x)\ u \quad \succ \quad u$$

$$\text{id}\ (\forall\alpha\ (\alpha{\rightarrow}\alpha))\ \text{id}\ (\forall\alpha\ (\alpha{\rightarrow}\alpha))\ \cdots\ \text{id}\ (\forall\alpha\ (\alpha{\rightarrow}\alpha))\ \text{id}\,B\,u \quad \succ^* \quad u$$

Examples

- **The polymorphic identity, again**

$$\mathrm{id}\ B\ u \quad \equiv \quad (\Lambda\alpha\,.\,\lambda x\!:\!\alpha\,.\,x)\ B\ u \quad \succ \quad (\lambda x\!:\!B\,.\,x)\ u \quad \succ \quad u$$

$$\mathrm{id}\ (\forall\alpha\ (\alpha{\to}\alpha))\ \mathrm{id}\ (\forall\alpha\ (\alpha{\to}\alpha))\ \cdots\ \mathrm{id}\ (\forall\alpha\ (\alpha{\to}\alpha))\ \mathrm{id}\, B\, u \quad \succ^* \quad u$$

- **A little bit more complex example. . .**

Examples

- **The polymorphic identity, again**

$$\mathrm{id}\ B\ u \quad \equiv \quad (\Lambda\alpha\,.\,\lambda x:\alpha\,.\,x)\ B\ u \quad \succ \quad (\lambda x:B\,.\,x)\ u \quad \succ \quad u$$

$$\mathrm{id}\ (\forall\alpha\ (\alpha{\to}\alpha))\ \mathrm{id}\ (\forall\alpha\ (\alpha{\to}\alpha))\ \cdots\ \mathrm{id}\ (\forall\alpha\ (\alpha{\to}\alpha))\ \mathrm{id}\, B\, u \quad \succ^* \quad u$$

- **A little bit more complex example...**

$$\begin{array}{l}(\Lambda\alpha\,.\,\lambda x:\alpha\,.\,\lambda f:\alpha{\to}\alpha\,.\,\overbrace{f\,(\cdots\,(f\ x)\cdots)}^{\text{32 times}}) \\ \quad (\forall\alpha\ (\alpha{\to}(\alpha{\to}\alpha){\to}\alpha))\ \ (\Lambda\alpha\,.\,\lambda x:\alpha\,.\,\lambda f:\alpha{\to}\alpha\,.\,f\ x) \\ \quad (\lambda n:\forall\alpha\ (\alpha{\to}(\alpha{\to}\alpha){\to}\alpha)\,.\,\Lambda\alpha\,.\,\lambda x:\alpha\,.\,\lambda f:\alpha{\to}\alpha\,.\,n\,\alpha\,(n\,\alpha\,x\,f)\,f)\end{array}$$

Examples

- **The polymorphic identity, again**

$$\text{id}\ B\ u \quad \equiv \quad (\Lambda\alpha\,.\,\lambda x : \alpha\,.\,x)\ B\ u \quad \succ \quad (\lambda x : B\,.\,x)\ u \quad \succ \quad u$$

$$\text{id}\ (\forall\alpha\ (\alpha\to\alpha))\ \text{id}\ (\forall\alpha\ (\alpha\to\alpha))\ \cdots\ \text{id}\ (\forall\alpha\ (\alpha\to\alpha))\ \text{id}\ B\ u \quad \succ^* \quad u$$

- **A little bit more complex example...**

$$\begin{array}{l}(\Lambda\alpha\,.\,\lambda x : \alpha\,.\,\lambda f : \alpha\to\alpha\,.\,\overbrace{f\ (\cdots\ (f\ x)}^{\text{32 times}}\cdots))\\ \quad(\forall\alpha\ (\alpha\to(\alpha\to\alpha)\to\alpha))\ \ (\Lambda\alpha\,.\,\lambda x : \alpha\,.\,\lambda f : \alpha\to\alpha\,.\,f\ x)\\ \quad(\lambda n : \forall\alpha\ (\alpha\to(\alpha\to\alpha)\to\alpha)\,.\,\Lambda\alpha\,.\,\lambda x : \alpha\,.\,\lambda f : \alpha\to\alpha\,.\,n\,\alpha\,(n\,\alpha\,x\,f)\,f)\end{array}$$

$$\succ^* \quad \Lambda\alpha\,.\,\lambda x : \alpha\,.\,\lambda f : \alpha\to\alpha\,.\,\underbrace{(f\ \cdots\ (f}_{\text{4 294 967 296 times}}\ x)\cdots)$$

Properties

Properties

Confluence

$$t \succ^* t_1 \;\wedge\; t \succ^* t_2 \quad \Rightarrow \quad \exists t' \; (t_1 \succ^* t' \;\wedge\; t_2 \succ^* t')$$

Properties

Confluence

$$t \succ^* t_1 \;\wedge\; t \succ^* t_2 \quad\Rightarrow\quad \exists t' \,(t_1 \succ^* t' \;\wedge\; t_2 \succ^* t')$$

Proof. Roughly the same as for the untyped λ-calculus (adaptation is easy)

Properties

Confluence

$$t \succ^* t_1 \ \wedge \ t \succ^* t_2 \quad \Rightarrow \quad \exists t' \ (t_1 \succ^* t' \ \wedge \ t_2 \succ^* t')$$

Proof. Roughly the same as for the untyped λ-calculus (adaptation is easy)

Church-Rosser

$$t_1 \simeq t_2 \quad \Leftrightarrow \quad \exists t' \ (t_1 \succ^* t' \ \wedge \ t_2 \succ^* t')$$

Properties

Confluence

$$t \succ^* t_1 \;\wedge\; t \succ^* t_2 \quad\Rightarrow\quad \exists t' \; (t_1 \succ^* t' \;\wedge\; t_2 \succ^* t')$$

Proof. Roughly the same as for the untyped λ-calculus (adaptation is easy)

Church-Rosser

$$t_1 \simeq t_2 \quad\Leftrightarrow\quad \exists t' \; (t_1 \succ^* t' \;\wedge\; t_2 \succ^* t')$$

Subject-reduction

If $\Gamma \vdash t : A$ and $t \succ^* t'$ then $\Gamma \vdash t : A$

Properties

Confluence

$$t \succ^* t_1 \;\wedge\; t \succ^* t_2 \quad \Rightarrow \quad \exists t' \,(t_1 \succ^* t' \;\wedge\; t_2 \succ^* t')$$

Proof. Roughly the same as for the untyped λ-calculus (adaptation is easy)

Church-Rosser

$$t_1 \simeq t_2 \quad \Leftrightarrow \quad \exists t' \,(t_1 \succ^* t' \;\wedge\; t_2 \succ^* t')$$

Subject-reduction

If $\Gamma \vdash t : A$ and $t \succ^* t'$ then $\Gamma \vdash t : A$

Proof. By induction on the derivation of $\Gamma \vdash t : A$, with $t \succ t'$ (one step reduction)

Properties

Confluence

$$t \succ^* t_1 \ \wedge \ t \succ^* t_2 \quad \Rightarrow \quad \exists t' \ (t_1 \succ^* t' \ \wedge \ t_2 \succ^* t')$$

Proof. Roughly the same as for the untyped λ-calculus (adaptation is easy)

Church-Rosser

$$t_1 \simeq t_2 \quad \Leftrightarrow \quad \exists t' \ (t_1 \succ^* t' \ \wedge \ t_2 \succ^* t')$$

Subject-reduction

If $\Gamma \vdash t : A$ and $t \succ^* t'$ then $\Gamma \vdash t : A$

Proof. By induction on the derivation of $\Gamma \vdash t : A$, with $t \succ t'$ (one step reduction)

Strong normalisation

All well-typed terms of system F are strongly normalisable

Properties

Confluence

$$t \succ^* t_1 \;\wedge\; t \succ^* t_2 \quad\Rightarrow\quad \exists t' \,(t_1 \succ^* t' \;\wedge\; t_2 \succ^* t')$$

Proof. Roughly the same as for the untyped λ-calculus (adaptation is easy)

Church-Rosser

$$t_1 \simeq t_2 \quad\Leftrightarrow\quad \exists t' \,(t_1 \succ^* t' \;\wedge\; t_2 \succ^* t')$$

Subject-reduction

If $\Gamma \vdash t : A$ and $t \succ^* t'$ then $\Gamma \vdash t : A$

Proof. By induction on the derivation of $\Gamma \vdash t : A$, with $t \succ t'$ (one step reduction)

Strong normalisation

All well-typed terms of system F are strongly normalisable

Proof. Girard and Tait's method of reducibility candidates (postponed)

Part II

Encoding data types

Booleans (1/3)

Booleans (1/3)

Encoding of booleans

$$\mathsf{Bool} \quad \equiv \quad \forall\gamma\ (\gamma \to \gamma \to \gamma)$$

Booleans (1/3)

Encoding of booleans

$$
\begin{aligned}
\mathsf{Bool} &\equiv \forall\gamma\,(\gamma \to \gamma \to \gamma) \\
\mathsf{true} &\equiv \Lambda\gamma\,.\,\lambda x, y : \gamma\,.\,x \quad : \quad \mathsf{Bool}
\end{aligned}
$$

Booleans (1/3)

Encoding of booleans

$$\begin{array}{lcl} \mathsf{Bool} & \equiv & \forall\gamma\ (\gamma \to \gamma \to \gamma) \\ \mathsf{true} & \equiv & \Lambda\gamma\,.\,\lambda x, y : \gamma\,.\,x \quad : \quad \mathsf{Bool} \\ \mathsf{false} & \equiv & \Lambda\gamma\,.\,\lambda x, y : \gamma\,.\,y \quad : \quad \mathsf{Bool} \end{array}$$

Booleans (1/3)

Encoding of booleans

$$\begin{array}{lcl} \mathsf{Bool} & \equiv & \forall\gamma\ (\gamma \to \gamma \to \gamma) \\ \mathsf{true} & \equiv & \Lambda\gamma\,.\,\lambda x, y : \gamma\,.\,x \quad : \quad \mathsf{Bool} \\ \mathsf{false} & \equiv & \Lambda\gamma\,.\,\lambda x, y : \gamma\,.\,y \quad : \quad \mathsf{Bool} \end{array}$$

$$\mathtt{if}_A\ u\ \mathtt{then}\ t_1\ \mathtt{else}\ t_2 \quad \equiv \quad u\ A\ t_1\ t_2$$

Booleans (1/3)

Encoding of booleans

$$\begin{array}{lcl} \mathsf{Bool} & \equiv & \forall\gamma\,(\gamma \to \gamma \to \gamma) \\ \mathsf{true} & \equiv & \Lambda\gamma\,.\,\lambda x, y : \gamma\,.\,x \quad : \quad \mathsf{Bool} \\ \mathsf{false} & \equiv & \Lambda\gamma\,.\,\lambda x, y : \gamma\,.\,y \quad : \quad \mathsf{Bool} \end{array}$$

$$\mathtt{if}_A \;\; u \;\; \mathtt{then} \;\; t_1 \;\; \mathtt{else} \;\; t_2 \quad \equiv \quad u \; A \; t_1 \; t_2$$

Correctness w.r.t. typing

Booleans (1/3)

Encoding of booleans

$$\begin{array}{lcl} \mathsf{Bool} & \equiv & \forall\gamma\,(\gamma \to \gamma \to \gamma) \\ \mathsf{true} & \equiv & \Lambda\gamma\,.\,\lambda x, y : \gamma\,.\,x \quad : \quad \mathsf{Bool} \\ \mathsf{false} & \equiv & \Lambda\gamma\,.\,\lambda x, y : \gamma\,.\,y \quad : \quad \mathsf{Bool} \end{array}$$

$$\mathtt{if}_A \; u \; \mathtt{then} \; t_1 \; \mathtt{else} \; t_2 \quad \equiv \quad u \; A \; t_1 \; t_2$$

Correctness w.r.t. typing

$$\frac{\Gamma \vdash u : \mathsf{Bool} \qquad \Gamma \vdash t_1 : A \qquad \Gamma \vdash t_2 : A}{\Gamma \vdash \mathtt{if}_A \; u \; \mathtt{then} \; t_1 \; \mathtt{else} \; t_2 : A}$$

Booleans (1/3)

Encoding of booleans

$$\begin{array}{lcl} \mathsf{Bool} & \equiv & \forall\gamma\,(\gamma \to \gamma \to \gamma) \\ \mathsf{true} & \equiv & \Lambda\gamma\,.\,\lambda x, y : \gamma\,.\,x \quad : \quad \mathsf{Bool} \\ \mathsf{false} & \equiv & \Lambda\gamma\,.\,\lambda x, y : \gamma\,.\,y \quad : \quad \mathsf{Bool} \end{array}$$

$$\mathtt{if}_A\ u\ \mathtt{then}\ t_1\ \mathtt{else}\ t_2 \quad \equiv \quad u\ A\ t_1\ t_2$$

Correctness w.r.t. typing

$$\frac{\Gamma \vdash u : \mathsf{Bool} \qquad \Gamma \vdash t_1 : A \qquad \Gamma \vdash t_2 : A}{\Gamma \vdash \mathtt{if}_A\ u\ \mathtt{then}\ t_1\ \mathtt{else}\ t_2 : A}$$

Correctness w.r.t. reduction

Booleans (1/3)

Encoding of booleans

$$\begin{array}{lcl} \mathsf{Bool} & \equiv & \forall\gamma\,(\gamma \to \gamma \to \gamma) \\ \mathsf{true} & \equiv & \Lambda\gamma\,.\,\lambda x, y : \gamma\,.\,x \quad : \quad \mathsf{Bool} \\ \mathsf{false} & \equiv & \Lambda\gamma\,.\,\lambda x, y : \gamma\,.\,y \quad : \quad \mathsf{Bool} \end{array}$$

$$\mathtt{if}_A\ u\ \mathtt{then}\ t_1\ \mathtt{else}\ t_2 \quad \equiv \quad u\ A\ t_1\ t_2$$

Correctness w.r.t. typing

$$\frac{\Gamma \vdash u : \mathsf{Bool} \qquad \Gamma \vdash t_1 : A \qquad \Gamma \vdash t_2 : A}{\Gamma \vdash \mathtt{if}_A\ u\ \mathtt{then}\ t_1\ \mathtt{else}\ t_2 : A}$$

Correctness w.r.t. reduction

$$\mathtt{if}_A\ \mathtt{true}\ \mathtt{then}\ t_1\ \mathtt{else}\ t_2 \quad \succ^* \quad t_1$$

Booleans (1/3)

Encoding of booleans

$$\mathsf{Bool} \equiv \forall\gamma\,(\gamma \to \gamma \to \gamma)$$

$$\begin{array}{lcll} \mathsf{true} & \equiv & \Lambda\gamma\,.\,\lambda x, y : \gamma\,.\,x & : \mathsf{Bool} \\ \mathsf{false} & \equiv & \Lambda\gamma\,.\,\lambda x, y : \gamma\,.\,y & : \mathsf{Bool} \end{array}$$

$$\mathtt{if}_A\ u\ \mathtt{then}\ t_1\ \mathtt{else}\ t_2 \equiv u\ A\ t_1\ t_2$$

Correctness w.r.t. typing

$$\frac{\Gamma \vdash u : \mathsf{Bool} \qquad \Gamma \vdash t_1 : A \qquad \Gamma \vdash t_2 : A}{\Gamma \vdash \mathtt{if}_A\ u\ \mathtt{then}\ t_1\ \mathtt{else}\ t_2 : A}$$

Correctness w.r.t. reduction

$$\begin{array}{lcl} \mathtt{if}_A\ \mathsf{true}\ \mathtt{then}\ t_1\ \mathtt{else}\ t_2 & \succ^* & t_1 \\ \mathtt{if}_A\ \mathsf{false}\ \mathtt{then}\ t_1\ \mathtt{else}\ t_2 & \succ^* & t_2 \end{array}$$

Booleans (2/3)

Objection:

Booleans (2/3)

Objection: We can do the same in the untyped λ-calculus!

Booleans (2/3)

Objection: We can do the same in the untyped λ-calculus!

$$\mathsf{true} \equiv \lambda x, y \,.\, x$$
$$\mathsf{false} \equiv \lambda x, y \,.\, y$$

$$\texttt{if}\ \ u\ \ \texttt{then}\ \ t_1\ \ \texttt{else}\ \ t_2 \ \equiv\ u\ t_1\ t_2$$

Booleans (2/3)

Objection: We can do the same in the untyped λ-calculus!

$$\left.\begin{array}{lcl} \text{true} & \equiv & \lambda x, y \,.\, x \\ \text{false} & \equiv & \lambda x, y \,.\, y \\ \texttt{if}\ u\ \texttt{then}\ t_1\ \texttt{else}\ t_2 & \equiv & u\ t_1\ t_2 \end{array}\right\}$$

Same reduction rules as before

Booleans (2/3)

Objection: We can do the same in the untyped λ-calculus!

$$\left.\begin{array}{lcl} \text{true} & \equiv & \lambda x, y \,.\, x \\ \text{false} & \equiv & \lambda x, y \,.\, y \\ \texttt{if}\ u\ \texttt{then}\ t_1\ \texttt{else}\ t_2 & \equiv & u\ t_1\ t_2 \end{array}\right\} \quad \text{Same reduction rules as before}$$

But nothing prevents the following computation:

$$\texttt{if}\ \underbrace{\lambda x \,.\, x}_{\text{bad bool}}\ \texttt{then}\ t_1\ \texttt{else}\ t_2 \ \equiv\ (\lambda x \,.\, x)\ t_1\ t_2 \ \succ\ \underbrace{t_1 t_2}_{\text{meaningless result}}$$

Booleans (2/3)

Objection: We can do the same in the untyped λ-calculus!

$$\left.\begin{array}{lcl} \text{true} & \equiv & \lambda x, y\,.\,x \\ \text{false} & \equiv & \lambda x, y\,.\,y \\ \texttt{if}\ u\ \texttt{then}\ t_1\ \texttt{else}\ t_2 & \equiv & u\ t_1\ t_2 \end{array}\right\}$$

Same reduction rules as before

But nothing prevents the following computation:

$$\texttt{if}\ \underbrace{\lambda x\,.\,x}_{\text{bad bool}}\ \texttt{then}\ t_1\ \texttt{else}\ t_2 \ \equiv\ (\lambda x\,.\,x)\ t_1\ t_2\ \succ\ \underbrace{t_1\,t_2}_{\text{meaningless result}}$$

Question: Does the type discipline of system F avoid this?

Booleans (3/3)

Principle (that should be satisfied by any functional programming language)

Booleans (3/3)

Principle (that should be satisfied by any functional programming language)

When a program P of type A evaluates to a value v, then v has one of the canonical forms expected by the type A.

Booleans (3/3)

Principle (that should be satisfied by any functional programming language)

When a program P of type A evaluates to a value v, then v has one of the canonical forms expected by the type A.

In ML/Haskell, a value produced by a program of type `Bool` will always be `true` or `false` (i.e. the canonical forms of type `bool`).

Booleans (3/3)

Principle (that should be satisfied by any functional programming language)

When a program P of type A evaluates to a value v, then v has one of the canonical forms expected by the type A.

In ML/Haskell, a value produced by a program of type `Bool` will always be `true` or `false` (i.e. the canonical forms of type `bool`).

In system F: Subject-reduction ensures that the normal form of a term of type Bool is a term of type Bool.

Booleans (3/3)

Principle (that should be satisfied by any functional programming language)

When a program P of type A evaluates to a value v, then v has one of the canonical forms expected by the type A.

In ML/Haskell, a value produced by a program of type `Bool` will always be `true` or `false` (i.e. the canonical forms of type `bool`).

In system F: Subject-reduction ensures that the normal form of a term of type Bool is a term of type Bool.

To conclude, it suffices to check that in system F:

Lemma (Canonical forms of type bool)

Booleans (3/3)

Principle (that should be satisfied by any functional programming language)

When a program P of type A evaluates to a value v, then v has one of the canonical forms expected by the type A.

In ML/Haskell, a value produced by a program of type `Bool` will always be `true` or `false` (i.e. the canonical forms of type `bool`).

In system F: Subject-reduction ensures that the normal form of a term of type Bool is a term of type Bool.

To conclude, it suffices to check that in system F:

Lemma (Canonical forms of type bool)

The terms $\text{true} \equiv \Lambda\gamma \,.\, \lambda x, y : \gamma \,.\, x$ and $\text{false} \equiv \Lambda\gamma \,.\, \lambda x, y : \gamma \,.\, y$ are the only closed normal terms of type $\text{Bool} \equiv \forall\gamma\ (\gamma\to\gamma\to\gamma)$

Booleans (3/3)

Principle (that should be satisfied by any functional programming language)

When a program P of type A evaluates to a value v, then v has one of the canonical forms expected by the type A.

In ML/Haskell, a value produced by a program of type `Bool` will always be `true` or `false` (i.e. the canonical forms of type `bool`).

In system F: Subject-reduction ensures that the normal form of a term of type Bool is a term of type Bool.

To conclude, it suffices to check that in system F:

Lemma (Canonical forms of type bool)

The terms $\text{true} \equiv \Lambda\gamma \,.\, \lambda x, y : \gamma \,.\, x$ and $\text{false} \equiv \Lambda\gamma \,.\, \lambda x, y : \gamma \,.\, y$ are the only closed normal terms of type $\text{Bool} \equiv \forall\gamma\ (\gamma\to\gamma\to\gamma)$

Proof. Case analysis on the derivation.

Cartesian product

Cartesian product

Encoding of the cartesian product $A \times B$

$$
\begin{array}{lcl}
A \times B & \equiv & \forall\gamma\ ((A{\to}B{\to}\gamma) \to \gamma) \\
\langle t_1\,, t_2\rangle & \equiv & \Lambda\gamma\,.\,\lambda f : A \to B \to \gamma\,.\, f\ \ t_1\ \ t_2 \\
\mathsf{fst} & \equiv & \lambda p : A \times B\,.\, p\ A\ (\lambda x : A\,.\,\lambda y : B\,.\,x) \quad : \quad A \times B \to A \\
\mathsf{snd} & \equiv & \lambda p : A \times B\,.\, p\ B\ (\lambda x : A\,.\,\lambda y : B\,.\,y) \quad : \quad A \times B \to B
\end{array}
$$

Cartesian product

Encoding of the cartesian product $A \times B$

$$
\begin{array}{lcl}
A \times B & \equiv & \forall\gamma\ ((A \to B \to \gamma) \to \gamma) \\
\langle t_1 , t_2 \rangle & \equiv & \Lambda\gamma\,.\,\lambda f : A \to B \to \gamma\,.\, f\ \ t_1\ \ t_2 \\
\text{fst} & \equiv & \lambda p : A \times B\,.\, p\ A\ (\lambda x : A\,.\,\lambda y : B\,.\,x) \quad : \quad A \times B \to A \\
\text{snd} & \equiv & \lambda p : A \times B\,.\, p\ B\ (\lambda x : A\,.\,\lambda y : B\,.\,y) \quad : \quad A \times B \to B
\end{array}
$$

Correctness w.r.t. typing and reduction

$$
\frac{\Gamma \vdash t_1 : A \qquad \Gamma \vdash t_2 : B}{\Gamma \vdash \langle t_1, t_2 \rangle : A \times B}
$$

$$
\begin{array}{lcl}
\text{fst}\ \langle t_1, t_2 \rangle & \succ^* & t_1 \\
\text{snd}\ \langle t_1, t_2 \rangle & \succ^* & t_2
\end{array}
$$

Cartesian product

Encoding of the cartesian product $A \times B$

$$
\begin{array}{lcl}
A \times B & \equiv & \forall\gamma\ ((A \to B \to \gamma) \to \gamma) \\
\langle t_1, t_2 \rangle & \equiv & \Lambda\gamma\,.\,\lambda f : A \to B \to \gamma\,.\,f\ t_1\ t_2 \\
\mathsf{fst} & \equiv & \lambda p : A \times B\,.\,p\ A\ (\lambda x : A\,.\,\lambda y : B\,.\,x) \quad : \quad A \times B \to A \\
\mathsf{snd} & \equiv & \lambda p : A \times B\,.\,p\ B\ (\lambda x : A\,.\,\lambda y : B\,.\,y) \quad : \quad A \times B \to B
\end{array}
$$

Correctness w.r.t. typing and reduction

$$
\frac{\Gamma \vdash t_1 : A \qquad \Gamma \vdash t_2 : B}{\Gamma \vdash \langle t_1, t_2 \rangle : A \times B}
\qquad\qquad
\begin{array}{lcl}
\mathsf{fst}\ \langle t_1, t_2 \rangle & \succ^* & t_1 \\
\mathsf{snd}\ \langle t_1, t_2 \rangle & \succ^* & t_2
\end{array}
$$

Lemma (Canonical forms of type $A \times B$)

The closed normal terms of type $A \times B$ are of the form $\langle t_1, t_2 \rangle$, where t_1 and t_2 are closed normal terms of type A and B, respectively.

Disjoint union

Disjoint union

Encoding of the disjoint union $A + B$

$$A + B \equiv \forall\gamma\ ((A{\to}\gamma) \to (B{\to}\gamma) \to \gamma)$$

$$\mathsf{inl}(v) \equiv \Lambda\gamma\,.\,\lambda f : A \to \gamma\,.\,\lambda g : B \to \gamma\,.\,f\ v \quad : \quad A + B \qquad (\text{with } v : A)$$

$$\mathsf{inr}(v) \equiv \Lambda\gamma\,.\,\lambda f : A \to \gamma\,.\,\lambda g : B \to \gamma\,.\,g\ v \quad : \quad A + B \qquad (\text{with } v : B)$$

$$\mathtt{case}_C\ u\ \mathtt{of}\ \ \mathsf{inl}(x) \mapsto t_1 \ \mid\ \mathsf{inr}(y) \mapsto t_2 \ \equiv\ u\ C\ (\lambda x : A\,.\,t_1)\ (\lambda y : B\,.\,t_2)$$

Disjoint union

Encoding of the disjoint union $A + B$

$$A + B \quad \equiv \quad \forall\gamma\ ((A{\to}\gamma) \to (B{\to}\gamma) \to \gamma)$$

$$\begin{array}{lclcll} \mathsf{inl}(v) & \equiv & \Lambda\gamma\,.\,\lambda f : A \to \gamma\,.\,\lambda g : B \to \gamma\,.\,f\ \ v & : & A + B & (\text{with } v : A) \\ \mathsf{inr}(v) & \equiv & \Lambda\gamma\,.\,\lambda f : A \to \gamma\,.\,\lambda g : B \to \gamma\,.\,g\ \ v & : & A + B & (\text{with } v : B) \end{array}$$

$$\mathtt{case}_C\ u\ \mathtt{of}\ \ \mathsf{inl}(x) \mapsto t_1\ \mid\ \mathsf{inr}(y) \mapsto t_2 \quad \equiv \quad u\ C\ (\lambda x : A\,.\,t_1)\ (\lambda y : B\,.\,t_2)$$

Correctness w.r.t. typing and reduction

$$\frac{\Gamma \vdash u : A + B \qquad \Gamma,\ x : A \vdash t_1 : C \qquad \Gamma,\ y : B \vdash t_2 : C}{\Gamma \ \vdash\ \mathtt{case}_C\ u\ \mathtt{of}\ \mathsf{inl}(x) \mapsto t_1\ \mid\ \mathsf{inr}(y) \mapsto t_2\ :\ C}$$

$$\begin{array}{lcl} \mathtt{case}_C\ \mathsf{inl}(v)\ \mathtt{of}\ \ \mathsf{inl}(x) \mapsto t_1\ \mid\ \mathsf{inr}(y) \mapsto t_2 & \succ^* & t_1\{x := v\} \\ \mathtt{case}_C\ \mathsf{inr}(v)\ \mathtt{of}\ \ \mathsf{inl}(x) \mapsto t_1\ \mid\ \mathsf{inr}(y) \mapsto t_2 & \succ^* & t_2\{y := v\} \end{array}$$

Disjoint union

Encoding of the disjoint union $A + B$

$$A + B \quad \equiv \quad \forall\gamma\ ((A{\to}\gamma) \to (B{\to}\gamma) \to \gamma)$$

$$\begin{array}{lcllll} \mathrm{inl}(v) & \equiv & \Lambda\gamma\,.\,\lambda f : A \to \gamma\,.\,\lambda g : B \to \gamma\,.\,f\ \ v & : & A + B & (\text{with } v : A) \\ \mathrm{inr}(v) & \equiv & \Lambda\gamma\,.\,\lambda f : A \to \gamma\,.\,\lambda g : B \to \gamma\,.\,g\ \ v & : & A + B & (\text{with } v : B) \end{array}$$

$$\mathtt{case}_C\ u\ \mathtt{of}\ \ \mathrm{inl}(x) \mapsto t_1\ \mid\ \mathrm{inr}(y) \mapsto t_2 \quad \equiv \quad u\ C\ (\lambda x : A\,.\,t_1)\ (\lambda y : B\,.\,t_2)$$

Correctness w.r.t. typing and reduction

$$\frac{\Gamma \vdash u : A + B \qquad \Gamma,\ x : A \vdash t_1 : C \qquad \Gamma,\ y : B \vdash t_2 : C}{\Gamma \ \vdash\ \mathtt{case}_C\ u\ \mathtt{of}\ \ \mathrm{inl}(x) \mapsto t_1\ \mid\ \mathrm{inr}(y) \mapsto t_2\ :\ C}$$

$$\begin{array}{lcl} \mathtt{case}_C\ \mathrm{inl}(v)\ \mathtt{of}\ \ \mathrm{inl}(x) \mapsto t_1\ \mid\ \mathrm{inr}(y) \mapsto t_2 & \succ^* & t_1\{x := v\} \\ \mathtt{case}_C\ \mathrm{inr}(v)\ \mathtt{of}\ \ \mathrm{inl}(x) \mapsto t_1\ \mid\ \mathrm{inr}(y) \mapsto t_2 & \succ^* & t_2\{y := v\} \end{array}$$

+ Canonical forms of type $A + B$ (works as expected)

Finite types

Finite types

Encoding of Fin_n ($n \geq 0$)

$$\mathsf{Fin}_n \equiv \forall\gamma\,(\underbrace{\gamma \to \cdots \to \gamma}_{n \text{ times}} \to \gamma)$$

$$\mathbf{e}_i \equiv \Lambda\gamma\,.\,\lambda x_1 : \gamma \ldots \lambda x_n : \gamma\,.\,x_i \quad : \quad \mathsf{Fin}_n \qquad (1 \leq i \leq n)$$

Finite types

Encoding of Fin_n ($n \geq 0$)

$$\mathsf{Fin}_n \equiv \forall\gamma\,(\underbrace{\gamma \to \cdots \to \gamma}_{n \text{ times}} \to \gamma)$$

$$\mathbf{e}_i \equiv \Lambda\gamma\,.\,\lambda x_1 : \gamma \ldots \lambda x_n : \gamma\,.\,x_i \quad : \quad \mathsf{Fin}_n \qquad (1 \leq i \leq n)$$

Again, $\mathbf{e}_1, \ldots, \mathbf{e}_n$ are the only closed normal terms of type Fin_n.

Finite types

Encoding of Fin_n ($n \geq 0$)

$$\mathsf{Fin}_n \quad \equiv \quad \forall\gamma\,(\underbrace{\gamma \to \cdots \to \gamma}_{n \text{ times}} \to \gamma)$$

$$\mathbf{e}_i \quad \equiv \quad \Lambda\gamma\,.\,\lambda x_1 : \gamma \ldots \lambda x_n : \gamma\,.\,x_i \quad : \quad \mathsf{Fin}_n \qquad (1 \leq i \leq n)$$

Again, $\mathbf{e}_1, \ldots, \mathbf{e}_n$ are the only closed normal terms of type Fin_n.

In particular:

$$\mathsf{Fin}_2 \quad \equiv \quad \forall\gamma\,(\gamma \to \gamma \to \gamma) \quad \equiv \quad \mathsf{Bool} \qquad \text{(type of booleans)}$$

Finite types

Encoding of Fin_n ($n \geq 0$)

$$\mathsf{Fin}_n \equiv \forall\gamma\,(\underbrace{\gamma \to \cdots \to \gamma}_{n \text{ times}} \to \gamma)$$

$$\mathbf{e}_i \equiv \Lambda\gamma\,.\,\lambda x_1 : \gamma \ldots \lambda x_n : \gamma\,.\,x_i \quad : \quad \mathsf{Fin}_n \qquad (1 \leq i \leq n)$$

Again, $\mathbf{e}_1, \ldots, \mathbf{e}_n$ are the only closed normal terms of type Fin_n.

In particular:

$$\mathsf{Fin}_2 \equiv \forall\gamma\,(\gamma \to \gamma \to \gamma) \equiv \mathsf{Bool} \qquad \text{(type of booleans)}$$

$$\mathsf{Fin}_1 \equiv \forall\gamma\,(\gamma \to \gamma) \equiv \mathsf{Unit} \qquad \text{(unit data-type)}$$

Finite types

Encoding of Fin_n $(n \geq 0)$

$$\mathsf{Fin}_n \equiv \forall\gamma\,(\underbrace{\gamma \to \cdots \to \gamma}_{n \text{ times}} \to \gamma)$$

$$\mathbf{e}_i \equiv \Lambda\gamma\,.\,\lambda x_1 : \gamma \ldots \lambda x_n : \gamma\,.\,x_i \quad : \quad \mathsf{Fin}_n \qquad (1 \leq i \leq n)$$

Again, $\mathbf{e}_1, \ldots, \mathbf{e}_n$ are the only closed normal terms of type Fin_n.

In particular:

Fin_2	$\equiv$	$\forall\gamma\,(\gamma \to \gamma \to \gamma)$	$\equiv$	Bool	(type of booleans)
Fin_1	$\equiv$	$\forall\gamma\,(\gamma \to \gamma)$	$\equiv$	Unit	(unit data-type)
Fin_0	$\equiv$	$\forall\gamma\,\gamma$	$\equiv$	$\bot$	(empty data-type)

Finite types

Encoding of Fin_n $(n \geq 0)$

$$\mathrm{Fin}_n \equiv \forall\gamma\,(\underbrace{\gamma \to \cdots \to \gamma}_{n \text{ times}} \to \gamma)$$

$$\mathbf{e}_i \equiv \Lambda\gamma\,.\,\lambda x_1 : \gamma \ldots \lambda x_n : \gamma\,.\,x_i \quad : \quad \mathrm{Fin}_n \qquad (1 \leq i \leq n)$$

Again, $\mathbf{e}_1, \ldots, \mathbf{e}_n$ are the only closed normal terms of type Fin_n.

In particular:

$$\mathrm{Fin}_2 \equiv \forall\gamma\,(\gamma \to \gamma \to \gamma) \equiv \mathrm{Bool}$$ (type of booleans)

$$\mathrm{Fin}_1 \equiv \forall\gamma\,(\gamma \to \gamma) \equiv \mathrm{Unit}$$ (unit data-type)

$$\mathrm{Fin}_0 \equiv \forall\gamma\,\gamma \equiv \bot$$ (empty data-type)

(Notice that there is no closed normal term of type $\bot$.)

Natural numbers

Natural numbers

Encoding of the type of Church numerals

$$\mathsf{Nat} \quad \equiv \quad \forall\gamma\ (\gamma \to (\gamma \to \gamma) \to \gamma)$$

Natural numbers

Encoding of the type of Church numerals

$$\mathsf{Nat} \equiv \forall\gamma\ (\gamma \to (\gamma \to \gamma) \to \gamma)$$

$$\begin{array}{lcl}
\overline{0} & \equiv & \Lambda\gamma\,.\,\lambda x:\gamma\,.\,\lambda f:\gamma{\to}\gamma\,.\,x \\
\overline{1} & \equiv & \Lambda\gamma\,.\,\lambda x:\gamma\,.\,\lambda f:\gamma{\to}\gamma\,.\,f\ \ x \\
\overline{2} & \equiv & \Lambda\gamma\,.\,\lambda x:\gamma\,.\,\lambda f:\gamma{\to}\gamma\,.\,f\ \ (f\ \ x) \\
 & \vdots & \\
\overline{n} & \equiv & \Lambda\gamma\,.\,\lambda x:\gamma\,.\,\lambda f:\gamma{\to}\gamma\,.\,\underbrace{f(\cdots(f}_{n\ \text{times}}\,x)\cdots) \quad : \quad \mathsf{Nat} \\
 & \vdots &
\end{array}$$

Natural numbers

Encoding of the type of Church numerals

$$\mathsf{Nat} \equiv \forall\gamma\,(\gamma \to (\gamma \to \gamma) \to \gamma)$$

$$\begin{array}{lcl}
\overline{0} & \equiv & \Lambda\gamma\,.\,\lambda x:\gamma\,.\,\lambda f:\gamma{\to}\gamma\,.\,x \\
\overline{1} & \equiv & \Lambda\gamma\,.\,\lambda x:\gamma\,.\,\lambda f:\gamma{\to}\gamma\,.\,f\ x \\
\overline{2} & \equiv & \Lambda\gamma\,.\,\lambda x:\gamma\,.\,\lambda f:\gamma{\to}\gamma\,.\,f\ (f\ x) \\
 & \vdots & \\
\overline{n} & \equiv & \Lambda\gamma\,.\,\lambda x:\gamma\,.\,\lambda f:\gamma{\to}\gamma\,.\,\underbrace{f(\cdots(f}_{n \text{ times}}\,x)\cdots) \quad : \quad \mathsf{Nat} \\
 & \vdots &
\end{array}$$

Lemma (Canonical forms of type Nat)

The terms $\overline{0}, \overline{1}, \overline{2}, \ldots$ are the only closed normal terms of type Nat.

Computing with natural numbers (1/2)

Computing with natural numbers (1/2)

Intuition: Church numeral $\overline{n}$ acts as an iterator:

$$\overline{n}\ A\ f\ x \quad \succ^* \quad \underbrace{f\ (\cdots(f}_{n}\ x)\cdots) \qquad (f : A \to A, \quad x : A)$$

Computing with natural numbers (1/2)

Intuition: Church numeral $\overline{n}$ acts as an iterator:

$$\overline{n}\ A\ f\ x \quad \succ^* \quad \underbrace{f\ (\cdots(f}_{n}\ x)\cdots) \qquad (f : A \to A, \quad x : A)$$

- Successor

$$\mathsf{succ} \quad \equiv \quad \lambda n : \mathsf{Nat}\,.\,\Lambda\gamma\,.\,\lambda x : \gamma\,.\,\lambda f : \gamma\to\gamma\,.\,f\ (n\ \gamma\ x\ f)$$

Computing with natural numbers (1/2)

Intuition: Church numeral $\overline{n}$ acts as an iterator:

$$\overline{n}\ A\ f\ x \quad \succ^* \quad \underbrace{f\ (\cdots(f}_{n}\ x)\cdots) \qquad (f : A \to A, \quad x : A)$$

- Successor

$$\mathsf{succ} \quad \equiv \quad \lambda n : \mathsf{Nat} \,.\, \Lambda\gamma \,.\, \lambda x : \gamma \,.\, \lambda f : \gamma{\to}\gamma \,.\, f\ (n\ \gamma\ x\ f)$$

- Addition

$$\mathsf{plus} \quad \equiv \quad \lambda n, m : \mathsf{Nat} \,.\, \Lambda\gamma \,.\, \lambda x : \gamma \,.\, \lambda f : \gamma{\to}\gamma \,.\, m\ \gamma\ (n\ \gamma\ x\ f)\ f$$

Computing with natural numbers (1/2)

Intuition: Church numeral $\overline{n}$ acts as an iterator:

$$\overline{n}\ A\ f\ x \quad \succ^* \quad \underbrace{f\ (\cdots(f}_{n}\ \ x)\cdots) \qquad (f : A \to A, \quad x : A)$$

- Successor

$$\mathrm{succ} \quad \equiv \quad \lambda n : \mathrm{Nat} \,.\, \Lambda\gamma \,.\, \lambda x : \gamma \,.\, \lambda f : \gamma{\to}\gamma \,.\, f\ (n\ \gamma\ x\ f)$$

- Addition

$$\begin{array}{lcl} \mathrm{plus} & \equiv & \lambda n, m : \mathrm{Nat} \,.\, \Lambda\gamma \,.\, \lambda x : \gamma \,.\, \lambda f : \gamma{\to}\gamma \,.\, m\ \gamma\ (n\ \gamma\ x\ f)\ f \\ \mathrm{plus}' & \equiv & \lambda n, m : \mathrm{Nat} \,.\, m\ \mathrm{Nat}\ n\ \mathrm{succ} \end{array}$$

Computing with natural numbers (1/2)

Intuition: Church numeral $\overline{n}$ acts as an iterator:

$$\overline{n}\ A\ f\ x \quad \succ^* \quad \underbrace{f\ (\cdots(f}_{n}\ x)\cdots) \qquad {\color{red}(f : A \to A, \quad x : A)}$$

- Successor

$$\mathsf{succ} \quad \equiv \quad \lambda n\,{:}\,\mathsf{Nat}\,.\,\Lambda\gamma\,.\,\lambda x\,{:}\,\gamma\,.\,\lambda f\,{:}\,\gamma{\to}\gamma\,.\,f\ (n\ \gamma\ x\ f)$$

- Addition

$$\begin{array}{lll} \mathsf{plus} & \equiv & \lambda n, m\,{:}\,\mathsf{Nat}\,.\,\Lambda\gamma\,.\,\lambda x\,{:}\,\gamma\,.\,\lambda f\,{:}\,\gamma{\to}\gamma\,.\,m\ \gamma\ (n\ \gamma\ x\ f)\ f \\ \mathsf{plus}' & \equiv & \lambda n, m\,{:}\,\mathsf{Nat}\,.\,m\ \mathsf{Nat}\ n\ \mathsf{succ} \end{array}$$

- Multiplication

$$\mathsf{mult} \quad \equiv \quad \lambda n, m\,{:}\,\mathsf{Nat}\,.\,\Lambda\gamma\,.\,\lambda x\,{:}\,\gamma\,.\,\lambda f\,{:}\,\gamma{\to}\gamma\,.\,n\ \gamma\ x\ (\lambda y\,{:}\,\gamma\,.\,m\ \gamma\ y\ f)$$

Computing with natural numbers (1/2)

Intuition: Church numeral $\overline{n}$ acts as an iterator:

$$\overline{n}\ A\ f\ x \quad \succ^* \quad \underbrace{f\ (\cdots (f}_{n}\ \ x)\cdots) \qquad (f : A \to A, \quad x : A)$$

- Successor

$$\mathsf{succ} \quad \equiv \quad \lambda n\!:\!\mathsf{Nat}\,.\,\Lambda\gamma\,.\,\lambda x\!:\!\gamma\,.\,\lambda f:\gamma\!\to\!\gamma\,.\,f\ (n\ \gamma\ x\ f)$$

- Addition

$$\begin{array}{lcl} \mathsf{plus} & \equiv & \lambda n, m\!:\!\mathsf{Nat}\,.\,\Lambda\gamma\,.\,\lambda x\!:\!\gamma\,.\,\lambda f:\gamma\!\to\!\gamma\,.\,m\ \gamma\ (n\ \gamma\ x\ f)\ f \\ \mathsf{plus}' & \equiv & \lambda n, m\!:\!\mathsf{Nat}\,.\,m\ \mathsf{Nat}\ n\ \mathsf{succ} \end{array}$$

- Multiplication

$$\begin{array}{lcl} \mathsf{mult} & \equiv & \lambda n, m\!:\!\mathsf{Nat}\,.\,\Lambda\gamma\,.\,\lambda x\!:\!\gamma\,.\,\lambda f:\gamma\!\to\!\gamma\,.\,n\ \gamma\ x\ (\lambda y\!:\!\gamma\,.\,m\ \gamma\ y\ f) \\ \mathsf{mult}' & \equiv & \lambda n, m\!:\!\mathsf{Nat}\,.\,n\ \mathsf{Nat}\ \overline{0}\ (\mathsf{plus}\ \ m) \end{array}$$

Computing with natural numbers (2/2)

Computing with natural numbers (2/2)

- Predecessor function $\quad \textsf{pred} : \textsf{Nat} \longrightarrow \textsf{Nat}$

Computing with natural numbers (2/2)

- Predecessor function $\text{pred} : \text{Nat} \to \text{Nat}$

$$\begin{array}{lcl} \text{pred } \overline{0} & \simeq & \overline{0} \\ \text{pred } (\overline{n+1}) & \simeq & \overline{n} \end{array}$$

Computing with natural numbers (2/2)

- Predecessor function $\color{red}{\mathsf{pred}} \;:\; \mathsf{Nat} \longrightarrow \mathsf{Nat}$

$$\begin{array}{lcl} \mathsf{pred}\ \overline{0} & \simeq & \overline{0} \\ \mathsf{pred}\ (\overline{n+1}) & \simeq & \overline{n} \end{array}$$

$$\begin{array}{lcll} \mathsf{fst} & \equiv & \lambda p : \mathsf{Nat}{\times}\mathsf{Nat} \,.\, p\ \mathsf{Nat}\ (\lambda x, y : \mathsf{Nat} \,.\, x) & : \quad \mathsf{Nat}{\times}\mathsf{Nat} \longrightarrow \mathsf{Nat} \\ \mathsf{snd} & \equiv & \lambda p : \mathsf{Nat}{\times}\mathsf{Nat} \,.\, p\ \mathsf{Nat}\ (\lambda x, y : \mathsf{Nat} \,.\, y) & : \quad \mathsf{Nat}{\times}\mathsf{Nat} \longrightarrow \mathsf{Nat} \\ \mathsf{step} & \equiv & \lambda p : \mathsf{Nat}{\times}\mathsf{Nat} \,.\, \langle \mathsf{snd}\ p,\ \mathsf{succ}\ (\mathsf{snd}\ p) \rangle & : \quad \mathsf{Nat}{\times}\mathsf{Nat} \longrightarrow \mathsf{Nat}{\times}\mathsf{Nat} \\ \mathsf{pred} & \equiv & \lambda n : \mathsf{Nat} \,.\, \mathsf{fst}\ (n\ (\mathsf{Nat}{\times}\mathsf{Nat})\ \langle \overline{0}, \overline{0} \rangle\ \mathsf{step}) & : \quad \mathsf{Nat} \longrightarrow \mathsf{Nat} \end{array}$$

Computing with natural numbers (2/2)

- Predecessor function $\text{pred} : \text{Nat} \longrightarrow \text{Nat}$

$$\begin{array}{lcl} \text{pred } \overline{0} & \simeq & \overline{0} \\ \text{pred } (\overline{n+1}) & \simeq & \overline{n} \end{array}$$

$$\begin{array}{llll} \text{fst} & \equiv & \lambda p:\text{Nat}\times\text{Nat}\,.\,p\ \text{Nat}\ (\lambda x, y:\text{Nat}\,.\,x) & : \quad \text{Nat}\times\text{Nat} \to \text{Nat} \\ \text{snd} & \equiv & \lambda p:\text{Nat}\times\text{Nat}\,.\,p\ \text{Nat}\ (\lambda x, y:\text{Nat}\,.\,y) & : \quad \text{Nat}\times\text{Nat} \to \text{Nat} \\ \text{step} & \equiv & \lambda p:\text{Nat}\times\text{Nat}\,.\,\langle \text{snd}\ p,\ \text{succ}\ (\text{snd}\ p)\rangle & : \quad \text{Nat}\times\text{Nat} \to \text{Nat}\times\text{Nat} \\ \text{pred} & \equiv & \lambda n:\text{Nat}\,.\,\text{fst}\ (n\ (\text{Nat}\times\text{Nat})\ \langle\overline{0},\overline{0}\rangle\ \text{step}) & : \quad \text{Nat} \to \text{Nat} \end{array}$$

- Ackerman function $\text{ack} : \text{Nat} \longrightarrow \text{Nat} \longrightarrow \text{Nat}$

Computing with natural numbers (2/2)

- Predecessor function $\textsf{pred} : \textsf{Nat} \to \textsf{Nat}$

$$\begin{array}{lcl} \textsf{pred}\ \overline{0} & \simeq & \overline{0} \\ \textsf{pred}\ (\overline{n+1}) & \simeq & \overline{n} \end{array}$$

$$\begin{array}{lclcl} \textsf{fst} & \equiv & \lambda p : \textsf{Nat}\times\textsf{Nat}\,.\, p\ \textsf{Nat}\ (\lambda x, y : \textsf{Nat}\,.\, x) & : & \textsf{Nat}\times\textsf{Nat} \to \textsf{Nat} \\ \textsf{snd} & \equiv & \lambda p : \textsf{Nat}\times\textsf{Nat}\,.\, p\ \textsf{Nat}\ (\lambda x, y : \textsf{Nat}\,.\, y) & : & \textsf{Nat}\times\textsf{Nat} \to \textsf{Nat} \\ \textsf{step} & \equiv & \lambda p : \textsf{Nat}\times\textsf{Nat}\,.\, \langle \textsf{snd}\ p,\ \textsf{succ}\ (\textsf{snd}\ p)\rangle & : & \textsf{Nat}\times\textsf{Nat} \to \textsf{Nat}\times\textsf{Nat} \\ \textsf{pred} & \equiv & \lambda n : \textsf{Nat}\,.\, \textsf{fst}\ (n\ (\textsf{Nat}\times\textsf{Nat})\ \langle\overline{0},\overline{0}\rangle\ \textsf{step}) & : & \textsf{Nat} \to \textsf{Nat} \end{array}$$

- Ackerman function $\textsf{ack} : \textsf{Nat} \to \textsf{Nat} \to \textsf{Nat}$

$$\begin{array}{llllll} \textsf{ack} & \overline{0} & \overline{m} & \simeq & \overline{m+1} \\ \textsf{ack} & (\overline{n+1}) & \overline{0} & \simeq & \textsf{ack}\ \overline{n}\ \overline{1} \\ \textsf{ack} & (\overline{n+1}) & (\overline{m+1}) & \simeq & \textsf{ack}\ \overline{n}\ (\textsf{ack}\ (\overline{n+1})\ \overline{m}) \end{array}$$

Computing with natural numbers (2/2)

- Predecessor function $\textsf{pred} : \textsf{Nat} \to \textsf{Nat}$

$$\begin{array}{lcl} \textsf{pred}\ \overline{0} & \simeq & \overline{0} \\ \textsf{pred}\ (\overline{n+1}) & \simeq & \overline{n} \end{array}$$

$$\begin{array}{lcl} \textsf{fst} & \equiv & \lambda p : \textsf{Nat}\times\textsf{Nat}\,.\ p\ \textsf{Nat}\ (\lambda x, y : \textsf{Nat}\,.\ x) & : & \textsf{Nat}\times\textsf{Nat} \to \textsf{Nat} \\ \textsf{snd} & \equiv & \lambda p : \textsf{Nat}\times\textsf{Nat}\,.\ p\ \textsf{Nat}\ (\lambda x, y : \textsf{Nat}\,.\ y) & : & \textsf{Nat}\times\textsf{Nat} \to \textsf{Nat} \\ \textsf{step} & \equiv & \lambda p : \textsf{Nat}\times\textsf{Nat}\,.\ \langle \textsf{snd}\ p,\ \textsf{succ}\ (\textsf{snd}\ p)\rangle & : & \textsf{Nat}\times\textsf{Nat} \to \textsf{Nat}\times\textsf{Nat} \\ \textsf{pred} & \equiv & \lambda n : \textsf{Nat}\,.\ \textsf{fst}\ (n\ (\textsf{Nat}\times\textsf{Nat})\ \langle\overline{0},\overline{0}\rangle\ \textsf{step}) & : & \textsf{Nat} \to \textsf{Nat} \end{array}$$

- Ackerman function $\textsf{ack} : \textsf{Nat} \to \textsf{Nat} \to \textsf{Nat}$

$$\begin{array}{lllcl} \textsf{ack} & \overline{0} & \overline{m} & \simeq & \overline{m+1} \\ \textsf{ack} & (\overline{n+1}) & \overline{0} & \simeq & \textsf{ack}\ \overline{n}\ \overline{1} \\ \textsf{ack} & (\overline{n+1}) & (\overline{m+1}) & \simeq & \textsf{ack}\ \overline{n}\ (\textsf{ack}\ (\overline{n+1})\ \overline{m}) \end{array}$$

$$\begin{array}{lcl} \textsf{down} & \equiv & \lambda f : (\textsf{Nat}{\to}\textsf{Nat})\,.\ \lambda p : \textsf{Nat}\,.\ p\ \textsf{Nat}\ (f\ \overline{1})\ f & : & (\textsf{Nat}{\to}\textsf{Nat}) \to (\textsf{Nat}{\to}\textsf{Nat}) \\ \textsf{ack} & \equiv & \lambda n, m : \textsf{Nat}\,.\ n\ (\textsf{Nat}{\to}\textsf{Nat})\ \textsf{succ}\ \textsf{down}\ m & : & \textsf{Nat} \to \textsf{Nat} \to \textsf{Nat} \end{array}$$

Computing with natural numbers (2/2)

- Predecessor function $\quad \text{pred} : \text{Nat} \to \text{Nat}$

$$\begin{array}{lcl} \text{pred } \overline{0} & \simeq & \overline{0} \\ \text{pred } (\overline{n+1}) & \simeq & \overline{n} \end{array}$$

$$\begin{array}{lcll} \text{fst} & \equiv & \lambda p : \text{Nat}\times\text{Nat} \,.\, p \text{ Nat } (\lambda x, y : \text{Nat} \,.\, x) & : \text{Nat}\times\text{Nat} \to \text{Nat} \\ \text{snd} & \equiv & \lambda p : \text{Nat}\times\text{Nat} \,.\, p \text{ Nat } (\lambda x, y : \text{Nat} \,.\, y) & : \text{Nat}\times\text{Nat} \to \text{Nat} \\ \text{step} & \equiv & \lambda p : \text{Nat}\times\text{Nat} \,.\, \langle \text{snd } p,\ \text{succ } (\text{snd } p) \rangle & : \text{Nat}\times\text{Nat} \to \text{Nat}\times\text{Nat} \\ \text{pred} & \equiv & \lambda n : \text{Nat} \,.\, \text{fst } (n \ (\text{Nat}\times\text{Nat})\ \langle \overline{0}, \overline{0} \rangle \text{ step}) & : \text{Nat} \to \text{Nat} \end{array}$$

- Ackerman function $\quad \text{ack} : \text{Nat} \to \text{Nat} \to \text{Nat}$

$$\begin{array}{llcl} \text{ack } \overline{0} & \overline{m} & \simeq & \overline{m+1} \\ \text{ack } (\overline{n+1}) & \overline{0} & \simeq & \text{ack } \overline{n}\ \overline{1} \\ \text{ack } (\overline{n+1}) & (\overline{m+1}) & \simeq & \text{ack } \overline{n}\ (\text{ack } (\overline{n+1})\ \overline{m}) \end{array}$$

$$\begin{array}{lcll} \text{down} & \equiv & \lambda f : (\text{Nat}\to\text{Nat}) \,.\, \lambda p : \text{Nat} \,.\, p \text{ Nat } (f\ \overline{1})\ f & : (\text{Nat}\to\text{Nat}) \to (\text{Nat}\to\text{Nat}) \\ \text{ack} & \equiv & \lambda n, m : \text{Nat} \,.\, n \ (\text{Nat}\to\text{Nat}) \text{ succ down } m & : \text{Nat} \to \text{Nat} \to \text{Nat} \end{array}$$

▷ SN theorem guarantees that all well-typed computations terminate

Part III

System F: Curry-style presentation

System F polymorphism

System F polymorphism

ML/Haskell polymorphism

Types	A, B	$::=$	$\alpha \mid A \to B \mid \cdots$ (user datatypes)
Schemes	S	$::=$	$\forall \vec{\alpha}\ B$

The type scheme $\forall \alpha\ B$ is defined after its particular instances $B\{\alpha := A\}$

System F polymorphism

ML/Haskell polymorphism

Types	A, B	$::=$	$\alpha \mid A \to B \mid \cdots$ (user datatypes)
Schemes	S	$::=$	$\forall \vec{\alpha}\ B$

The type scheme $\forall \alpha\ B$ is defined after its particular instances $B\{\alpha := A\}$
$\Rightarrow$ Type system is predicative

System F polymorphism

ML/Haskell polymorphism

Types	A, B	$::=$	$\alpha \mid A \rightarrow B \mid \cdots$ (user datatypes)
Schemes	S	$::=$	$\forall\vec{\alpha}\ B$

The type scheme $\forall\alpha\ B$ is defined after its particular instances $B\{\alpha := A\}$
$\Rightarrow$ Type system is predicative

System F polymorphism

Types	A, B	$::=$	$\alpha \mid A \rightarrow B \mid \forall\alpha\ B$

The type $\forall\alpha\ B$ and its instances $B\{\alpha{:=}A\}$ are defined simultaneously

System F polymorphism

ML/Haskell polymorphism

Types	A, B	$::=$	$\alpha \mid A \to B \mid \cdots$ (user datatypes)
Schemes	S	$::=$	$\forall \vec{\alpha}\ B$

The type scheme $\forall \alpha\ B$ is defined after its particular instances $B\{\alpha := A\}$
$\Rightarrow$ Type system is predicative

System F polymorphism

Types	A, B	$::=$	$\alpha \mid A \to B \mid \forall \alpha\ B$

The type $\forall \alpha\ B$ and its instances $B\{\alpha{:=}A\}$ are defined simultaneously

$$\forall \alpha\ (\alpha \to \alpha) \qquad \text{and} \qquad \forall \alpha\ (\alpha \to \alpha) \to \forall \alpha\ (\alpha \to \alpha)$$

System F polymorphism

ML/Haskell polymorphism

Types	A, B	$::=$	$\alpha \mid A \to B \mid \cdots$ (user datatypes)
Schemes	S	$::=$	$\forall \vec{\alpha}\ B$

The type scheme $\forall \alpha\ B$ is defined after its particular instances $B\{\alpha := A\}$
$\Rightarrow$ Type system is predicative

System F polymorphism

Types	A, B	$::=$	$\alpha \mid A \to B \mid \forall \alpha\ B$

The type $\forall \alpha\ B$ and its instances $B\{\alpha := A\}$ are defined simultaneously

$$\forall \alpha\ (\alpha \to \alpha) \qquad \text{and} \qquad \forall \alpha\ (\alpha \to \alpha) \to \forall \alpha\ (\alpha \to \alpha)$$

$\Rightarrow$ Type system is impredicative, or cyclic

Extracting pure λ-terms

Extracting pure λ-terms

In Church-style system F, polymorphism is explicit:

$$\mathrm{id} \equiv \Lambda\alpha\,.\,\lambda x : \alpha\,.\,x \qquad \text{and} \qquad \mathrm{id}\ \mathrm{Nat}\ 2$$

- Two kind of redexes $\quad (\lambda x : A\,.\,t)u \quad$ and $\quad (\Lambda\alpha\,.\,t)A$

Extracting pure λ-terms

In Church-style system F, polymorphism is explicit:

$$\text{id} \;\equiv\; \Lambda\alpha\,.\,\lambda x : \alpha\,.\,x \qquad \text{and} \qquad \text{id Nat } 2$$

- Two kind of redexes $\quad (\lambda x : A\,.\,t)u \quad$ and $\quad (\Lambda\alpha\,.\,t)A$

Idea: Remove type abstractions/applications/annotations

Extracting pure λ-terms

In Church-style system F, polymorphism is explicit:

$$\mathrm{id} \equiv \Lambda\alpha\,.\,\lambda x:\alpha\,.\,x \qquad \text{and} \qquad \mathrm{id}\ \mathrm{Nat}\ 2$$

- Two kind of redexes $\quad (\lambda x:A\,.\,t)u \quad$ and $\quad (\Lambda\alpha\,.\,t)A$

Idea: Remove type abstractions/applications/annotations

Erasing function $\quad t \mapsto |t|$

$$\begin{array}{rcl@{\qquad\qquad}rcl} |x| & = & x & & & \\ |\lambda x:A\,.\,t| & = & \lambda x\,.\,|t| & |\Lambda\alpha\,.\,t| & = & |t| \\ |tu| & = & |t||u| & |tA| & = & |t| \end{array}$$

Extracting pure λ-terms

In Church-style system F, polymorphism is explicit:

$$\mathrm{id} \equiv \Lambda\alpha\,.\,\lambda x : \alpha\,.\,x \qquad \text{and} \qquad \mathrm{id}\ \mathsf{Nat}\ 2$$

- Two kind of redexes $\quad (\lambda x : A\,.\,t)u \quad$ and $\quad (\Lambda\alpha\,.\,t)A$

Idea: Remove type abstractions/applications/annotations

Erasing function $\quad t \mapsto |t|$

$$\begin{array}{rcl} |x| & = & x \\ |\lambda x : A\,.\,t| & = & \lambda x\,.\,|t| \\ |tu| & = & |t||u| \end{array} \qquad \begin{array}{rcl} |\Lambda\alpha\,.\,t| & = & |t| \\ |tA| & = & |t| \end{array}$$

- Target language is pure λ-calculus
- Second kind redexes are erased, first kind redexes are preserved

Extending the erasing function

Erased terms have a nice computational behaviour...

- Only one kind of redex, easy to execute (Krivine's machine)
- Irrelevant part of computation has been removed
- The essence of computation has been preserved (to be justified later)

Extending the erasing function

Erased terms have a nice computational behaviour. . .

- Only one kind of redex, easy to execute (Krivine's machine)
- Irrelevant part of computation has been removed
- The essence of computation has been preserved (to be justified later)

. . . but what is their status w.r.t. typing?

Extending the erasing function

Erased terms have a nice computational behaviour...

- Only one kind of redex, easy to execute (Krivine's machine)
- Irrelevant part of computation has been removed
- The essence of computation has been preserved (to be justified later)

... but what is their status w.r.t. typing?

The erasing function, defined on terms, can be extended to:

Extending the erasing function

Erased terms have a nice computational behaviour...

- Only one kind of redex, easy to execute (Krivine's machine)
- Irrelevant part of computation has been removed
- The essence of computation has been preserved (to be justified later)

... but what is their status w.r.t. typing?

The erasing function, defined on terms, can be extended to:

- The whole syntax

Extending the erasing function

Erased terms have a nice computational behaviour. . .

- Only one kind of redex, easy to execute (Krivine's machine)
- Irrelevant part of computation has been removed
- The essence of computation has been preserved (to be justified later)

. . . but what is their status w.r.t. typing?

The erasing function, defined on terms, can be extended to:

- The whole syntax
- The judgements

Extending the erasing function

Erased terms have a nice computational behaviour...

- Only one kind of redex, easy to execute (Krivine's machine)
- Irrelevant part of computation has been removed
- The essence of computation has been preserved (to be justified later)

... but what is their status w.r.t. typing?

The erasing function, defined on terms, can be extended to:

- The whole syntax
- The judgements
- The typing rules

Extending the erasing function

Erased terms have a nice computational behaviour. . .

- Only one kind of redex, easy to execute (Krivine's machine)
- Irrelevant part of computation has been removed
- The essence of computation has been preserved (to be justified later)

. . . but what is their status w.r.t. typing?

The erasing function, defined on terms, can be extended to:

- The whole syntax
- The judgements
- The typing rules
- The derivations

Extending the erasing function

Erased terms have a nice computational behaviour...

- Only one kind of redex, easy to execute (Krivine's machine)
- Irrelevant part of computation has been removed
- The essence of computation has been preserved (to be justified later)

... but what is their status w.r.t. typing?

The erasing function, defined on terms, can be extended to:

- The whole syntax
- The judgements
- The typing rules
- The derivations

$\Rightarrow$ Induces a new formalism: Curry-style system F

Church-style system F

Types	A, B	$::=$	$\alpha \mid A \to B \mid \forall\alpha\ B$
Terms	t, u	$::=$	$x \mid \lambda x : A\,.\,t \mid tu \mid \Lambda\alpha\,.\,t \mid tA$
Judgments	Γ	$::=$	$[] \mid \Gamma,\ x{:}A$

Reduction

$$(\lambda x : A\,.\,t)u \succ t\{x := u\}$$
$$(\Lambda\alpha\,.\,t)A \succ t\{\alpha := A\}$$

Church-style system F

Types	A, B	$::=$	$\alpha \mid A \to B \mid \forall\alpha\, B$
Terms	t, u	$::=$	$x \mid \lambda x : A \,.\, t \mid tu \mid \Lambda\alpha \,.\, t \mid tA$
Judgments	Γ	$::=$	$[] \mid \Gamma,\ x{:}A$

Reduction

$$(\lambda x : A \,.\, t)u \succ t\{x := u\}$$
$$(\Lambda\alpha \,.\, t)A \succ t\{\alpha := A\}$$

Curry-style system F [Leivant 83]

Types	A, B	$::=$	$\alpha \mid A \to B \mid \forall\alpha\ B$
Terms	t, u	$::=$	$x \mid \lambda x\,.\,t \mid tu$
Judgments	Γ	$::=$	$[] \mid \Gamma,\ x{:}A$

Reduction $(\lambda x\,.\,t)u \succ t\{x := u\}$

Curry-style system F [Leivant 83]

Types	A, B	$::=$	$\alpha \mid A \to B \mid \forall\alpha\ B$
Terms	t, u	$::=$	$x \mid \lambda x\,.\,t \mid tu$
Judgments	Γ	$::=$	$[] \mid \Gamma,\ x{:}A$

Reduction $$(\lambda x\,.\,t)u \succ t\{x := u\}$$

Remarks:

- Types (and contexts) are unchanged
- Terms are now pure λ-terms
- Only one kind of redex

Church-style system F: typing rules

$$\frac{}{\Gamma \vdash x : A}\ (x{:}A)\in\Gamma$$

$$\frac{\Gamma,\ x : A \vdash t : B}{\Gamma \vdash \lambda x : A\,.\,t : A \to B} \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : B}$$

$$\frac{\Gamma \vdash t : B}{\Gamma \vdash \Lambda\alpha\,.\,t : \forall\alpha\ B}\ \alpha\notin TV(\Gamma) \qquad \frac{\Gamma \vdash t : \forall\alpha\ B}{\Gamma \vdash tA : B\{\alpha := A\}}$$

Curry-style system F: typing rules

$$\frac{}{\Gamma \vdash x : A} \; (x{:}A) \in \Gamma$$

$$\frac{\Gamma,\; x : A \vdash t : B}{\Gamma \vdash \lambda x \,.\, t : A \to B} \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : B}$$

$$\frac{\Gamma \vdash t : B}{\Gamma \vdash t : \forall\alpha\; B} \; \alpha \notin TV(\Gamma) \qquad \frac{\Gamma \vdash t : \forall\alpha\; B}{\Gamma \vdash t : B\{\alpha := A\}}$$

Curry-style system F: typing rules

$$\frac{}{\Gamma \vdash x : A} \quad (x{:}A) \in \Gamma$$

$$\frac{\Gamma,\ x : A \vdash t : B}{\Gamma \vdash \lambda x \,.\, t : A \rightarrow B} \qquad \frac{\Gamma \vdash t : A \rightarrow B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : B}$$

$$\frac{\Gamma \vdash t : B}{\Gamma \vdash t : \forall\alpha\ B} \quad \alpha \notin TV(\Gamma) \qquad \frac{\Gamma \vdash t : \forall\alpha\ B}{\Gamma \vdash t : B\{\alpha := A\}}$$

$\Rightarrow$ Rules are no more syntax directed

Curry-style system F: properties

Things that do not change

Curry-style system F: properties

Things that do not change

- Substitutivity + β-subject reduction

Curry-style system F: properties

Things that do not change

- Substitutivity + β-subject reduction
- Strong normalisation (postponed)

Curry-style system F: properties

Things that do not change

- Substitutivity + β-subject reduction
- Strong normalisation (postponed)

Things that change

Curry-style system F: properties

Things that do not change

- Substitutivity + β-subject reduction
- Strong normalisation (postponed)

Things that change

- A term may have several types

$$\Delta \equiv \lambda x \,.\, x\ x$$

Curry-style system F: properties

Things that do not change

- Substitutivity + β-subject reduction
- Strong normalisation (postponed)

Things that change

- A term may have several types

$$\Delta \equiv \lambda x\,.\,x\ x \quad : \quad \forall\alpha\,(\alpha \to \alpha) \ \to\ \forall\alpha\,(\alpha \to \alpha)$$

Curry-style system F: properties

Things that do not change

- Substitutivity + β-subject reduction
- Strong normalisation (postponed)

Things that change

- A term may have several types

$$\begin{aligned}\Delta \equiv \lambda x \,.\, x\ x \quad &: \quad \forall\alpha\,(\alpha \to \alpha) \ \to \ \forall\alpha\,(\alpha \to \alpha) \\ &: \quad \forall\alpha\ \alpha \ \to \ \forall\alpha\ \alpha\end{aligned}$$

Curry-style system F: properties

Things that do not change

- Substitutivity + β-subject reduction
- Strong normalisation (postponed)

Things that change

- A term may have several types

$$\begin{array}{lll} \Delta \equiv \lambda x\,.\,x\;x & : & \forall\alpha\,(\alpha\to\alpha) \;\to\; \forall\alpha\,(\alpha\to\alpha) \\ & : & \forall\alpha\;\alpha \;\to\; \forall\alpha\;\alpha \\ & : & \forall\alpha\;\alpha \;\to\; \forall\alpha\,(\alpha\to\alpha) \end{array}$$

Curry-style system F: properties

Things that do not change

- Substitutivity + β-subject reduction
- Strong normalisation (postponed)

Things that change

- A term may have several types

$$
\begin{array}{rcl}
\Delta \equiv \lambda x \,.\, x\ x & : & \forall\alpha\,(\alpha\to\alpha) \;\to\; \forall\alpha\,(\alpha\to\alpha) \\
& : & \forall\alpha\ \alpha \;\to\; \forall\alpha\ \alpha \\
& : & \forall\alpha\ \alpha \;\to\; \forall\alpha\,(\alpha\to\alpha) \\
& : & \mathsf{Bool}\to\mathsf{Bool}\to\mathsf{Bool}
\end{array}
$$

Curry-style system F: properties

Things that do not change

- Substitutivity + β-subject reduction
- Strong normalisation (postponed)

Things that change

- A term may have several types

$$
\begin{array}{rcl}
\Delta \equiv \lambda x \,.\, x\; x & : & \forall\alpha\,(\alpha\to\alpha) \;\to\; \forall\alpha\,(\alpha\to\alpha) \\
 & : & \forall\alpha\;\alpha \;\to\; \forall\alpha\;\alpha \\
 & : & \forall\alpha\;\alpha \;\to\; \forall\alpha\,(\alpha\to\alpha) \\
 & : & \mathsf{Bool}\to\mathsf{Bool}\to\mathsf{Bool} \qquad (\text{`or' function!})
\end{array}
$$

Curry-style system F: properties

Things that do not change

- Substitutivity + β-subject reduction
- Strong normalisation (postponed)

Things that change

- A term may have several types

$$
\begin{array}{rcll}
\Delta \equiv \lambda x\,.\,x\;x & : & \forall\alpha\,(\alpha\to\alpha) \;\to\; \forall\alpha\,(\alpha\to\alpha) & \\
& : & \forall\alpha\,\alpha \;\to\; \forall\alpha\,\alpha & \\
& : & \forall\alpha\,\alpha \;\to\; \forall\alpha\,(\alpha\to\alpha) & \\
& : & \mathsf{Bool}\to\mathsf{Bool}\to\mathsf{Bool} & \text{('or' function!)}
\end{array}
$$

- No principal type (cf later)

Curry-style system F: properties

Things that do not change

- Substitutivity + β-subject reduction
- Strong normalisation (postponed)

Things that change

- A term may have several types

$$
\begin{array}{rcll}
\Delta \equiv \lambda x . x\ x & : & \forall\alpha\,(\alpha \to \alpha) \ \to\ \forall\alpha\,(\alpha \to \alpha) & \\
 & : & \forall\alpha\ \alpha \ \to\ \forall\alpha\ \alpha & \\
 & : & \forall\alpha\ \alpha \ \to\ \forall\alpha\,(\alpha \to \alpha) & \\
 & : & \mathsf{Bool} \to \mathsf{Bool} \to \mathsf{Bool} & \text{('or' function!)}
\end{array}
$$

- No principal type (cf later)
- Type checking/inference becomes undecidable [Wells 94]

Erasing and typing

Erasing and typing

Equivalence between Church and Curry's presentations

Erasing and typing

Equivalence between Church and Curry's presentations

1. If $\Gamma \vdash t_0 : A$ (Church), then $\Gamma \vdash |t_0| : A$ (Curry)

Erasing and typing

Equivalence between Church and Curry's presentations

1. If $\Gamma \vdash t_0 : A$ (Church), then $\Gamma \vdash |t_0| : A$ (Curry)
2. If $\Gamma \vdash t : A$ (Curry), then $\Gamma \vdash t_0 : A$ (Church) for some t_0 s.t. $|t_0| = t$

Erasing and typing

Equivalence between Church and Curry's presentations

1. If $\Gamma \vdash t_0 : A$ (Church), then $\Gamma \vdash |t_0| : A$ (Curry)
2. If $\Gamma \vdash t : A$ (Curry), then $\Gamma \vdash t_0 : A$ (Church) for some t_0 s.t. $|t_0| = t$

The erasing function maps:

Church's world	Curry's world

Erasing and typing

Equivalence between Church and Curry's presentations

1. If $\Gamma \vdash t_0 : A$ (Church), then $\Gamma \vdash |t_0| : A$ (Curry)
2. If $\Gamma \vdash t : A$ (Curry), then $\Gamma \vdash t_0 : A$ (Church) for some t_0 s.t. $|t_0| = t$

The erasing function maps:

	Church's world		**Curry's world**	
1.	derivations	to	derivations	(isomorphism)

Erasing and typing

Equivalence between Church and Curry's presentations

1. If $\Gamma \vdash t_0 : A$ (Church), then $\Gamma \vdash |t_0| : A$ (Curry)
2. If $\Gamma \vdash t : A$ (Curry), then $\Gamma \vdash t_0 : A$ (Church) for some t_0 s.t. $|t_0| = t$

The erasing function maps:

	Church's world		Curry's world	
1.	derivations	to	derivations	(isomorphism)
2.	valid judgements	to	valid judgements	(surjective only)

Erasing and typing

Equivalence between Church and Curry's presentations

1. If $\Gamma \vdash t_0 : A$ (Church), then $\Gamma \vdash |t_0| : A$ (Curry)
2. If $\Gamma \vdash t : A$ (Curry), then $\Gamma \vdash t_0 : A$ (Church) for some t_0 s.t. $|t_0| = t$

The erasing function maps:

	Church's world		Curry's world	
1.	derivations	to	derivations	(isomorphism)
2.	valid judgements	to	valid judgements	(surjective only)

On valid judgements, erasing is not injective:

$$\begin{array}{rcl}
\lambda f : (\forall\alpha\ (\alpha\to\alpha))\,.\,f(\forall\alpha\ (\alpha\to\alpha))f & : & \forall\alpha\ (\alpha\to\alpha)\ \to\ \forall\alpha\ (\alpha\to\alpha) \\
\lambda f : (\forall\alpha\ (\alpha\to\alpha))\,.\,\Lambda\alpha\,.\,f(\alpha\to\alpha)(f\alpha) & : & \forall\alpha\ (\alpha\to\alpha)\ \to\ \forall\alpha\ (\alpha\to\alpha) \\
\rightsquigarrow \quad \lambda f\,.\,ff & : & \forall\alpha\ (\alpha\to\alpha)\ \to\ \forall\alpha\ (\alpha\to\alpha)
\end{array}$$

Erasing and reduction

Second-kind redexes are erased, first-kind redexes are preserved

Erasing and reduction

Second-kind redexes are erased, first-kind redexes are preserved

(Church) $$(\Lambda\alpha\,.\,\lambda x:\alpha\,.\,x)\,B\,y \quad \succ \quad (\lambda x:B\,.\,x)\,y \quad \succ \quad y$$

Erasing and reduction

Second-kind redexes are erased, first-kind redexes are preserved

$$
\begin{array}{llcccc}
\text{(Church)} & (\Lambda\alpha\,.\,\lambda x:\alpha\,.\,x)\,B\,y & \succ & (\lambda x:B\,.\,x)\,y & \succ & y \\
\quad\downarrow \text{ Erasing} & & & & & \\
\text{(Curry)} & (\lambda x\,.\,x)\,y & \equiv & (\lambda x\,.\,x)\,y & \succ & y
\end{array}
$$

Erasing and reduction

Second-kind redexes are erased, first-kind redexes are preserved

$$\begin{array}{lccccc} \text{(Church)} & (\Lambda\alpha\,.\,\lambda x:\alpha\,.\,x)\,B\,y & \succ & (\lambda x:B\,.\,x)\,y & \succ & y \\ \quad\downarrow \text{ Erasing} & & & & & \\ \text{(Curry)} & (\lambda x\,.\,x)\,y & \equiv & (\lambda x\,.\,x)\,y & \succ & y \end{array}$$

Fact 1 (Church to Curry):

If $t_0, t_0' \in \text{Church}$, then

$$t \succ^n t' \quad \Rightarrow \quad |t_0| \succ^p |t_0'| \qquad (\text{with } p \leq n)$$

Erasing and reduction

Second-kind redexes are erased, first-kind redexes are preserved

$$
\begin{array}{lccccc}
\text{(Church)} & (\Lambda\alpha\,.\,\lambda x:\alpha\,.\,x)\,B\,y & \succ & (\lambda x:B\,.\,x)\,y & \succ & y \\
\quad\downarrow \text{Erasing} & & & & & \\
\text{(Curry)} & (\lambda x\,.\,x)\,y & \equiv & (\lambda x\,.\,x)\,y & \succ & y
\end{array}
$$

Fact 1 (Church to Curry):

If $t_0, t_0' \in \text{Church}$, then

$$t \succ^n t' \quad \Rightarrow \quad |t_0| \succ^p |t_0'| \qquad \text{(with } p \leq n\text{)}$$

Fact 2 (Curry to Church):

If $t_0 \in \text{Church}$, $t' \in \text{Curry}$ and t_0 well-typed, then

$$|t_0| \succ^p t' \quad \Rightarrow \quad \exists t_0' \ (|t_0'| = t' \ \wedge \ t_0 \succ^n t_0') \qquad \text{(with } n \geq p\text{)}$$

Normalisation equivalence

Normalisation equivalence

Fact 3 (Combinatorial argument):

Normalisation equivalence

Fact 3 (Combinatorial argument):

1. During the contraction of a 1st-kind redex, the number of redexes of both kinds may increase

Normalisation equivalence

Fact 3 (Combinatorial argument):

1. During the contraction of a 1st-kind redex, the number of redexes of both kinds may increase
2. During the contraction of a 2nd-kind redex

Normalisation equivalence

Fact 3 (Combinatorial argument):

1. During the contraction of a 1st-kind redex, the number of redexes of both kinds may increase
2. During the contraction of a 2nd-kind redex
    - the number of 1st-kind redexes may increase

Normalisation equivalence

Fact 3 (Combinatorial argument):

1. During the contraction of a 1st-kind redex, the number of redexes of both kinds may increase
2. During the contraction of a 2nd-kind redex
   - the number of 1st-kind redexes may increase
   - the number of 2nd-kind redexes does not increase

Normalisation equivalence

Fact 3 (Combinatorial argument):

1. During the contraction of a 1st-kind redex, the number of redexes of both kinds may increase
2. During the contraction of a 2nd-kind redex
   - the number of 1st-kind redexes may increase
   - the number of 2nd-kind redexes does not increase
   - the number of type abstractions $(\Lambda\alpha \,.\, t)$ decreases

Normalisation equivalence

Fact 3 (Combinatorial argument):

1. During the contraction of a 1st-kind redex, the number of redexes of both kinds may increase
2. During the contraction of a 2nd-kind redex
   - the number of 1st-kind redexes may increase
   - the number of 2nd-kind redexes does not increase
   - the number of type abstractions $(\Lambda\alpha \,.\, t)$ decreases

Combining facts 1, 2 and 3, we easily prove:

Theorem (Normalisation equivalence):

The following statements are combinatorially equivalent:

1. All typable terms of syst. F-Church are strongly normalisable
2. All typable terms of syst. F-Curry are strongly normalisable

Subtyping

Subtyping

In Curry-style system F, subtyping is introduced as a macro:

$$A \leq B \quad \equiv \quad x : A \vdash x : B$$

Subtyping

In Curry-style system F, subtyping is introduced as a macro:

$$A \leq B \quad \equiv \quad x : A \vdash x : B$$

Admissible rules

Subtyping

In Curry-style system F, subtyping is introduced as a macro:

$$A \leq B \quad \equiv \quad x : A \vdash x : B$$

Admissible rules

(Reflexivity, transitivity)

$$\frac{}{A \leq A} \qquad \frac{A \leq B \qquad B \leq C}{A \leq C}$$

Subtyping

In Curry-style system F, subtyping is introduced as a macro:

$$A \leq B \quad \equiv \quad x : A \vdash x : B$$

Admissible rules

(Reflexivity, transitivity)

$$\frac{}{A \leq A} \qquad \frac{A \leq B \qquad B \leq C}{A \leq C}$$

(Polymorphism)

$$\frac{}{\forall\alpha\, B \leq B\{\alpha := A\}} \qquad \frac{A \leq B}{A \leq \forall\alpha\, B}\ \alpha \notin TV(A)$$

Subtyping

In Curry-style system F, subtyping is introduced as a macro:

$$A \leq B \quad \equiv \quad x : A \vdash x : B$$

Admissible rules

(Reflexivity, transitivity)

$$\frac{}{A \leq A} \qquad \frac{A \leq B \qquad B \leq C}{A \leq C}$$

(Polymorphism)

$$\frac{}{\forall\alpha\ B \leq B\{\alpha := A\}} \qquad \frac{A \leq B}{A \leq \forall\alpha\ B}\ \alpha \notin TV(A)$$

(Subsumption)

$$\frac{\Gamma \vdash t : A \qquad A \leq B}{\Gamma \vdash t : B}$$

Problem with η-redexes in Curry-style system F

Problem with η-redexes in Curry-style system F

- The (desired) subtyping rule for arrow-types

$$\frac{A \leq A' \qquad B \leq B'}{A' \to B \ \leq \ A \to B'}$$

is not admissible

Problem with η-redexes in Curry-style system F

- The (desired) subtyping rule for arrow-types

$$\frac{A \leq A' \qquad B \leq B'}{A' \to B \ \leq \ A \to B'}$$

is not admissible

- In particular, we have: $\quad f : \mathsf{Nat} \to \forall\beta\, \beta \ \not\vdash \ f \ : \ \forall\alpha\, \alpha \to \mathsf{Bool}$

Problem with η-redexes in Curry-style system F

- The (desired) subtyping rule for arrow-types

$$\frac{A \leq A' \qquad B \leq B'}{A' \rightarrow B \ \leq \ A \rightarrow B'}$$

  is not admissible

- In particular, we have: $f : \mathsf{Nat} \rightarrow \forall\beta\ \beta \ \nvdash \ f \ : \ \forall\alpha\ \alpha \rightarrow \mathsf{Bool}$

  but if we η-expand: $f : \mathsf{Nat} \rightarrow \forall\beta\ \beta \ \vdash \ \lambda x\,.\,fx \ : \ \forall\alpha\ \alpha \rightarrow \mathsf{Bool}$

Problem with η-redexes in Curry-style system F

- The (desired) subtyping rule for arrow-types

$$\frac{A \leq A' \qquad B \leq B'}{A' \to B \;\leq\; A \to B'}$$

is not admissible

- In particular, we have: $f : \mathsf{Nat} \to \forall\beta\,\beta \;\not\vdash\; f \;:\; \forall\alpha\,\alpha \to \mathsf{Bool}$

  but if we η-expand: $f : \mathsf{Nat} \to \forall\beta\,\beta \;\vdash\; \lambda x\,.\,fx \;:\; \forall\alpha\,\alpha \to \mathsf{Bool}$

- This shows that:

Problem with η-redexes in Curry-style system F

- The (desired) subtyping rule for arrow-types

$$\frac{A \leq A' \qquad B \leq B'}{A' \to B \ \leq \ A \to B'}$$

  is not admissible

- In particular, we have: $f : \mathsf{Nat} \to \forall\beta\ \beta \ \nvdash \ f \ : \ \forall\alpha\ \alpha \to \mathsf{Bool}$

  but if we η-expand: $f : \mathsf{Nat} \to \forall\beta\ \beta \ \vdash \ \lambda x\,.\,fx \ : \ \forall\alpha\ \alpha \to \mathsf{Bool}$

- This shows that:
  1. Curry-style system F does not enjoy η-subject reduction

Problem with η-redexes in Curry-style system F

- The (desired) subtyping rule for arrow-types

$$\frac{A \leq A' \qquad B \leq B'}{A' \to B \ \leq \ A \to B'}$$

  is not admissible

- In particular, we have: $f : \mathsf{Nat} \to \forall\beta\ \beta \ \nvdash \ f \ : \ \forall\alpha\ \alpha \to \mathsf{Bool}$

  but if we η-expand: $f : \mathsf{Nat} \to \forall\beta\ \beta \ \vdash \ \lambda x\,.\,fx \ : \ \forall\alpha\ \alpha \to \mathsf{Bool}$

- This shows that:
  1. Curry-style system F does not enjoy η-subject reduction
  2. This problem is connected with subtyping in arrow-types

Problem with η-redexes in Curry-style system F

- The (desired) subtyping rule for arrow-types

$$\frac{A \leq A' \qquad B \leq B'}{A' \to B \;\leq\; A \to B'}$$

is not admissible

- In particular, we have: $f : \mathrm{Nat} \to \forall\beta\ \beta \not\vdash f : \forall\alpha\ \alpha \to \mathrm{Bool}$

  but if we η-expand: $f : \mathrm{Nat} \to \forall\beta\ \beta \vdash \lambda x\,.\,fx : \forall\alpha\ \alpha \to \mathrm{Bool}$

- This shows that:
  1. Curry-style system F does not enjoy η-subject reduction
  2. This problem is connected with subtyping in arrow-types

The well-typed term: $\lambda x\,.\,fx \;:\; (\forall\alpha\ \alpha) \to \mathrm{Bool}$ (Curry-style)

comes from the term $\underbrace{\lambda x : (\forall\alpha\ \alpha)\,.\,f\ (x\ \mathrm{Nat})\ \mathrm{Bool}}_{\text{not an } \eta\text{-redex}}$ (Church-style)

System F_η [Mitchell 88]

System F_η [Mitchell 88]

Extend Curry-style system F with a new rule

$$\frac{\Gamma \vdash \lambda x\,.\,tx : A}{\Gamma \vdash t : A}\ {\scriptstyle x \notin FV(t)}$$

to enforce η-subject reduction

System F_η [Mitchell 88]

Extend Curry-style system F with a new rule

$$\frac{\Gamma \vdash \lambda x\,.\,tx : A}{\Gamma \vdash t : A}\ {\scriptstyle x \notin FV(t)}$$

to enforce η-subject reduction

Properties:

System F_η [Mitchell 88]

Extend Curry-style system F with a new rule

$$\frac{\Gamma \vdash \lambda x\,.\,tx : A}{\Gamma \vdash t : A}\ x{\notin}FV(t)$$

to enforce η-subject reduction

Properties:

- Substitutivity, $\beta\eta$-subject-reduction, strong normalisation

System F_η [Mitchell 88]

Extend Curry-style system F with a new rule

$$\frac{\Gamma \vdash \lambda x\,.\,tx : A}{\Gamma \vdash t : A}\ {\scriptstyle x \notin FV(t)}$$

to enforce η-subject reduction

Properties:

- Substitutivity, $\beta\eta$-subject-reduction, strong normalisation
- Subtyping rule $\dfrac{A \leq A' \qquad B \leq B'}{A' \rightarrow B \ \leq\ A \rightarrow B'}$ is now admissible

System F_η [Mitchell 88]

Extend Curry-style system F with a new rule

$$\frac{\Gamma \vdash \lambda x\,.\,tx : A}{\Gamma \vdash t : A}\ {\scriptstyle x \notin FV(t)}$$

to enforce η-subject reduction

Properties:

- Substitutivity, $\beta\eta$-subject-reduction, strong normalisation
- Subtyping rule $\dfrac{A \leq A' \qquad B \leq B'}{A' \to B \;\leq\; A \to B'}$ is now admissible

Expansion lemma

If $\Gamma \vdash t : A$ is derivable in F_η, then $\Gamma \vdash t' : A$ is derivable in system F for some η-expansion t' of the term t.

More subtyping

If we set

$$\begin{array}{rcl} \bot & := & \forall\gamma\ \gamma \\ A \times B & := & \forall\gamma\ ((A \to B \to \gamma) \to \gamma) \\ A + B & := & \forall\gamma\ ((A \to \gamma) \to (B \to \gamma) \to \gamma) \\ \mathrm{List}(A) & := & \forall\gamma\ (\gamma \to (A \to \gamma \to \gamma) \to \gamma) \end{array}$$

then, in F_η, the following subtyping rules are admissible:

$$\frac{}{\bot\ \leq\ A} \qquad \frac{A\ \leq\ A'}{\mathrm{List}(A)\ \leq\ \mathrm{List}(A')}$$

$$\frac{A\ \leq\ A' \qquad B\ \leq\ B'}{A \times B\ \leq\ A' \times B'} \qquad \frac{A\ \leq\ A' \qquad B\ \leq\ B'}{A + B\ \leq\ A' + B'}$$

More subtyping

If we set

$$\begin{array}{rcl} \bot & := & \forall\gamma\ \gamma \\ A \times B & := & \forall\gamma\ ((A \to B \to \gamma) \to \gamma) \\ A + B & := & \forall\gamma\ ((A \to \gamma) \to (B \to \gamma) \to \gamma) \\ \mathrm{List}(A) & := & \forall\gamma\ (\gamma \to (A \to \gamma \to \gamma) \to \gamma) \end{array}$$

then, in F_η, the following subtyping rules are admissible:

$$\frac{}{\bot \ \leq \ A} \qquad \frac{A \ \leq \ A'}{\mathrm{List}(A) \ \leq \ \mathrm{List}(A')}$$

$$\frac{A \ \leq \ A' \qquad B \ \leq \ B'}{A \times B \ \leq \ A' \times B'} \qquad \frac{A \ \leq \ A' \qquad B \ \leq \ B'}{A + B \ \leq \ A' + B'}$$

But most typable terms have no principal type

Adding intersection types

Extend system F_η with binary intersections

$$\textbf{Types} \quad A, B \quad ::= \quad \alpha \quad | \quad A \to B \quad | \quad \forall \alpha\ B \quad | \quad A \cap B$$

Adding intersection types

Extend system F_η with binary intersections

$$\textbf{Types} \quad A, B \quad ::= \quad \alpha \quad | \quad A \to B \quad | \quad \forall\alpha\ B \quad | \quad A \cap B$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash t : B}{\Gamma \vdash t : A \cap B} \qquad \frac{\Gamma \vdash t : A \cap B}{\Gamma \vdash t : A} \qquad \frac{\Gamma \vdash t : A \cap B}{\Gamma \vdash t : B}$$

Adding intersection types

Extend system F_η with binary intersections

$$\textbf{Types} \quad A, B \quad ::= \quad \alpha \quad | \quad A \to B \quad | \quad \forall\alpha\, B \quad | \quad A \cap B$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash t : B}{\Gamma \vdash t : A \cap B} \qquad \frac{\Gamma \vdash t : A \cap B}{\Gamma \vdash t : A} \qquad \frac{\Gamma \vdash t : A \cap B}{\Gamma \vdash t : B}$$

- $\beta\eta$-subject reduction, strong normalisation, etc.

Adding intersection types

Extend system F_η with binary intersections

$$\textbf{Types} \quad A, B \quad ::= \quad \alpha \quad | \quad A \rightarrow B \quad | \quad \forall\alpha\ B \quad | \quad A \cap B$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash t : B}{\Gamma \vdash t : A \cap B} \qquad \frac{\Gamma \vdash t : A \cap B}{\Gamma \vdash t : A} \qquad \frac{\Gamma \vdash t : A \cap B}{\Gamma \vdash t : B}$$

- $\beta\eta$-subject reduction, strong normalisation, etc.
- Subtyping rules

$$\frac{}{A \cap B \ \leq \ A} \qquad \frac{}{A \cap B \ \leq \ B} \qquad \frac{C \leq A \qquad C \leq B}{C \ \leq \ A \cap B}$$

Adding intersection types

Extend system F_η with binary intersections

$$\textbf{Types} \quad A, B \quad ::= \quad \alpha \quad | \quad A \to B \quad | \quad \forall\alpha\, B \quad | \quad A \cap B$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash t : B}{\Gamma \vdash t : A \cap B} \qquad \frac{\Gamma \vdash t : A \cap B}{\Gamma \vdash t : A} \qquad \frac{\Gamma \vdash t : A \cap B}{\Gamma \vdash t : B}$$

- $\beta\eta$-subject reduction, strong normalisation, etc.
- Subtyping rules

$$\frac{}{A \cap B \;\leq\; A} \qquad \frac{}{A \cap B \;\leq\; B} \qquad \frac{C \leq A \qquad C \leq B}{C \;\leq\; A \cap B}$$

- All the strongly normalising terms are typable. . .

Adding intersection types

Extend system F_η with binary intersections

Types $\quad A, B \quad ::= \quad \alpha \quad | \quad A \to B \quad | \quad \forall\alpha\, B \quad | \quad A \cap B$

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash t : B}{\Gamma \vdash t : A \cap B} \qquad \frac{\Gamma \vdash t : A \cap B}{\Gamma \vdash t : A} \qquad \frac{\Gamma \vdash t : A \cap B}{\Gamma \vdash t : B}$$

- $\beta\eta$-subject reduction, strong normalisation, etc.
- Subtyping rules

$$\frac{}{A \cap B \ \leq \ A} \qquad \frac{}{A \cap B \ \leq \ B} \qquad \frac{C \leq A \qquad C \leq B}{C \ \leq \ A \cap B}$$

- All the strongly normalising terms are typable...

  ... but nothing to do with $\forall$: already true in $\lambda{\to}\cap$

Adding intersection types

Extend system F_η with binary intersections

Types $\quad A, B \quad ::= \quad \alpha \quad | \quad A \to B \quad | \quad \forall\alpha\, B \quad | \quad A \cap B$

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash t : B}{\Gamma \vdash t : A \cap B} \qquad \frac{\Gamma \vdash t : A \cap B}{\Gamma \vdash t : A} \qquad \frac{\Gamma \vdash t : A \cap B}{\Gamma \vdash t : B}$$

- $\beta\eta$-subject reduction, strong normalisation, etc.
- Subtyping rules

$$\frac{}{A \cap B \;\leq\; A} \qquad \frac{}{A \cap B \;\leq\; B} \qquad \frac{C \leq A \qquad C \leq B}{C \;\leq\; A \cap B}$$

- All the strongly normalising terms are typable. . .

  . . . but nothing to do with $\forall$: already true in $\lambda{\to}\cap$
- All typable terms have a principal type

  $\lambda x : xx\,. \quad : \quad \forall\alpha\, \forall\beta\, ((\alpha{\to}\beta) \cap \alpha \to \beta)$

Part IV

The Strong Normalisation Theorem

The meaning of second-order quantification (1/2)

Question: What is the meaning of $\forall\alpha\ (\alpha \to \alpha)$?

The meaning of second-order quantification (1/2)

Question: What is the meaning of $\forall\alpha\ (\alpha \to \alpha)$?

First scenario: an infinite Cartesian product (à la Martin-Löf)

$$\forall\alpha\ (\alpha \to \alpha) \quad \approx \quad \prod_{\alpha \text{ type}} (\alpha \to \alpha)$$

The meaning of second-order quantification (1/2)

Question: What is the meaning of $\forall\alpha\ (\alpha \to \alpha)$?

First scenario: an infinite Cartesian product (à la Martin-Löf)

$$\begin{aligned}\forall\alpha\ (\alpha \to \alpha) \ &\approx\ \prod_{\alpha\ \text{type}} (\alpha \to \alpha) \\ &\approx\ (\bot \to \bot) \times (\mathsf{Bool} \to \mathsf{Bool}) \times (\mathsf{Nat} \to \mathsf{Nat}) \times \cdots\end{aligned}$$

The meaning of second-order quantification (1/2)

Question: What is the meaning of $\forall\alpha\ (\alpha \to \alpha)$?

First scenario: an infinite Cartesian product (à la Martin-Löf)

$$\forall\alpha\ (\alpha \to \alpha) \approx \prod_{\alpha\ \text{type}} (\alpha \to \alpha)$$

$$\approx (\bot \to \bot) \times (\mathsf{Bool} \to \mathsf{Bool}) \times (\mathsf{Nat} \to \mathsf{Nat}) \times \cdots$$

Since all the types $A \to A$ are inhabited:

The meaning of second-order quantification (1/2)

Question: What is the meaning of $\forall\alpha\ (\alpha \to \alpha)$?

First scenario: an infinite Cartesian product (à la Martin-Löf)

$$\forall\alpha\ (\alpha \to \alpha) \approx \prod_{\alpha \text{ type}} (\alpha \to \alpha)$$

$$\approx (\bot \to \bot) \times (\mathsf{Bool} \to \mathsf{Bool}) \times (\mathsf{Nat} \to \mathsf{Nat}) \times \cdots$$

Since all the types $A \to A$ are inhabited:

1. The cartesian product $\forall\alpha\ (\alpha{\to}\alpha)$ should be larger than all the types of the form $A \to A$

The meaning of second-order quantification (1/2)

Question: What is the meaning of $\forall\alpha\ (\alpha \to \alpha)$?

First scenario: an infinite Cartesian product (à la Martin-Löf)

$$\forall\alpha\ (\alpha \to \alpha) \approx \prod_{\alpha \text{ type}} (\alpha \to \alpha)$$

$$\approx (\bot \to \bot) \times (\mathsf{Bool} \to \mathsf{Bool}) \times (\mathsf{Nat} \to \mathsf{Nat}) \times \cdots$$

Since all the types $A \to A$ are inhabited:

1. The cartesian product $\forall\alpha\ (\alpha{\to}\alpha)$ should be larger than all the types of the form $A \to A$
2. In particular, $\forall\alpha\ (\alpha{\to}\alpha)$ should be larger than its own function space $\forall\alpha\ (\alpha{\to}\alpha) \to \forall\alpha\ (\alpha{\to}\alpha)\ldots$

The meaning of second-order quantification (1/2)

Question: What is the meaning of $\forall\alpha\ (\alpha \to \alpha)$?

First scenario: an infinite Cartesian product (à la Martin-Löf)

$$\forall\alpha\ (\alpha \to \alpha) \approx \prod_{\alpha\ \text{type}} (\alpha \to \alpha)$$
$$\approx (\bot \to \bot) \times (\text{Bool} \to \text{Bool}) \times (\text{Nat} \to \text{Nat}) \times \cdots$$

Since all the types $A \to A$ are inhabited:

1. The cartesian product $\forall\alpha\ (\alpha{\to}\alpha)$ should be larger than all the types of the form $A \to A$
2. In particular, $\forall\alpha\ (\alpha{\to}\alpha)$ should be larger than its own function space $\forall\alpha\ (\alpha{\to}\alpha) \to \forall\alpha\ (\alpha{\to}\alpha)\ldots$

. . . seems to be very confusing!

The meaning of second-order quantification (2/2)

Second scenario: In F-Curry, both rules $\forall$-intro and $\forall$-elim

$$\frac{\Gamma \vdash t : B}{\Gamma \vdash t : \forall\alpha\ B}\ \alpha \notin TV(\Gamma) \qquad\qquad \frac{\Gamma \vdash t : \forall\alpha\ B}{\Gamma \vdash t : B\{\alpha := A\}}$$

suggest that $\forall$ is not a cartesian product, but an intersection

The meaning of second-order quantification (2/2)

Second scenario: In F-Curry, both rules $\forall$-intro and $\forall$-elim

$$\frac{\Gamma \vdash t : B}{\Gamma \vdash t : \forall\alpha\ B}\ \alpha \notin TV(\Gamma) \qquad\qquad \frac{\Gamma \vdash t : \forall\alpha\ B}{\Gamma \vdash t : B\{\alpha := A\}}$$

suggest that $\forall$ is not a cartesian product, but an intersection

Taking back our example:

The meaning of second-order quantification (2/2)

Second scenario: In F-Curry, both rules $\forall$-intro and $\forall$-elim

$$\frac{\Gamma \vdash t : B}{\Gamma \vdash t : \forall\alpha\ B}\ \alpha \notin TV(\Gamma) \qquad\qquad \frac{\Gamma \vdash t : \forall\alpha\ B}{\Gamma \vdash t : B\{\alpha := A\}}$$

suggest that $\forall$ is not a cartesian product, but an intersection

Taking back our example:

1. The intersection $\forall\alpha\ (\alpha \to \alpha)$ is smaller than all $A \to A$

The meaning of second-order quantification (2/2)

Second scenario: In F-Curry, both rules $\forall$-intro and $\forall$-elim

$$\frac{\Gamma \vdash t : B}{\Gamma \vdash t : \forall\alpha\ B}\ \alpha \notin TV(\Gamma) \qquad\qquad \frac{\Gamma \vdash t : \forall\alpha\ B}{\Gamma \vdash t : B\{\alpha := A\}}$$

suggest that $\forall$ is not a cartesian product, but an intersection

Taking back our example:

1. The intersection $\forall\alpha\ (\alpha \to \alpha)$ is smaller than all $A \to A$
2. In particular, $\forall\alpha\ (\alpha \to \alpha)$ is smaller than its own function space $\forall\alpha\ (\alpha \to \alpha) \to \forall\alpha\ (\alpha \to \alpha) \ldots$

The meaning of second-order quantification (2/2)

Second scenario: In F-Curry, both rules ∀-intro and ∀-elim

$$\frac{\Gamma \vdash t : B}{\Gamma \vdash t : \forall\alpha\ B}\ \alpha \notin TV(\Gamma) \qquad\qquad \frac{\Gamma \vdash t : \forall\alpha\ B}{\Gamma \vdash t : B\{\alpha := A\}}$$

suggest that $\forall$ is not a cartesian product, but an intersection

Taking back our example:

1. The intersection $\forall\alpha\ (\alpha\to\alpha)$ is smaller than all $A \to A$
2. In particular, $\forall\alpha\ (\alpha\to\alpha)$ is smaller than its own function space $\forall\alpha\ (\alpha\to\alpha) \to \forall\alpha\ (\alpha\to\alpha)\ldots$

... our intuition feels much better!

The meaning of second-order quantification (2/2)

Second scenario: In F-Curry, both rules $\forall$-intro and $\forall$-elim

$$\frac{\Gamma \vdash t : B}{\Gamma \vdash t : \forall\alpha\ B}\ \alpha \notin TV(\Gamma) \qquad\qquad \frac{\Gamma \vdash t : \forall\alpha\ B}{\Gamma \vdash t : B\{\alpha := A\}}$$

suggest that $\forall$ is not a cartesian product, but an intersection

Taking back our example:

1. The <u>intersection</u> $\forall\alpha\ (\alpha \to \alpha)$ is smaller than all $A \to A$
2. In particular, $\forall\alpha\ (\alpha \to \alpha)$ is smaller than its own function space $\forall\alpha\ (\alpha \to \alpha) \to \forall\alpha\ (\alpha \to \alpha) \ldots$

... our intuition feels much better!

$\Rightarrow$ We will prove strong normalisation for Curry-style system F

Remember that SN(F-Church) $\Leftrightarrow$ SN(F-Curry) (combinatorial equivalence)

Strong normalisation: the difficulty

Try to prove that

$$\Gamma \vdash t : A \quad \Rightarrow \quad t \text{ is SN}$$

by induction on the derivation of $\Gamma \vdash t : A$

Strong normalisation: the difficulty

Try to prove that

$$\Gamma \vdash t : A \quad \Rightarrow \quad t \text{ is SN}$$

by induction on the derivation of $\Gamma \vdash t : A$

$$\frac{}{\Gamma \vdash x : A}\ (x{:}A)\in\Gamma$$

$$\frac{\Gamma,\ x : A \vdash t : B}{\Gamma \vdash \lambda x\,.\,t : A \to B} \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : B}$$

$$\frac{\Gamma \vdash t : B}{\Gamma \vdash t : \forall\alpha\ B}\ \alpha\notin TV(\Gamma) \qquad \frac{\Gamma \vdash t : \forall\alpha\ B}{\Gamma \vdash t : B\{\alpha := A\}}$$

Strong normalisation: the difficulty

Try to prove that

$$\Gamma \vdash t : A \quad \Rightarrow \quad t \text{ is SN}$$

by induction on the derivation of $\Gamma \vdash t : A$

$$\frac{}{\Gamma \vdash x : A}\ {\scriptstyle (x:A)\in\Gamma}$$

$$\frac{\Gamma,\ x : A \vdash t : B}{\Gamma \vdash \lambda x\,.\,t : A \to B} \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : B}$$

$$\frac{\Gamma \vdash t : B}{\Gamma \vdash t : \forall\alpha\ B}\ {\scriptstyle \alpha\notin TV(\Gamma)} \qquad \frac{\Gamma \vdash t : \forall\alpha\ B}{\Gamma \vdash t : B\{\alpha := A\}}$$

All the cases successfully pass the test except application

Two terms t and u may be SN, whereas tu is not [Take $t \equiv u \equiv \lambda x\,.\,xx$]

Strong normalisation: the difficulty

Try to prove that

$$\Gamma \vdash t : A \quad \Rightarrow \quad t \text{ is SN}$$

by induction on the derivation of $\Gamma \vdash t : A$

$$\frac{}{\Gamma \vdash x : A}\ {\scriptstyle (x:A)\in\Gamma}$$

$$\frac{\Gamma,\ x : A \vdash t : B}{\Gamma \vdash \lambda x\,.\,t : A \to B} \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : B}$$

$$\frac{\Gamma \vdash t : B}{\Gamma \vdash t : \forall\alpha\ B}\ {\scriptstyle \alpha\notin TV(\Gamma)} \qquad \frac{\Gamma \vdash t : \forall\alpha\ B}{\Gamma \vdash t : B\{\alpha := A\}}$$

All the cases successfully pass the test except application

Two terms t and u may be SN, whereas tu is not [Take $t \equiv u \equiv \lambda x\,.\,xx$]

$\Rightarrow$ The induction hypothesis "t is SN" is too weak (in general)

Reducibility candidates [Girard 1971]

To prove that

$$\Gamma \vdash t : A \quad \Rightarrow \quad t \text{ is SN},$$

the induction hypothesis "t is SN" is too weak.

Reducibility candidates [Girard 1971]

To prove that

$$\Gamma \vdash t : A \quad \Rightarrow \quad t \text{ is SN},$$

the induction hypothesis "t is SN" is too weak.

$\Rightarrow$ Should replace it by an invariant that depends on the type A

Reducibility candidates [Girard 1971]

To prove that

$$\Gamma \vdash t : A \quad \Rightarrow \quad t \text{ is SN},$$

the induction hypothesis "t is SN" is too weak.

$\Rightarrow$ Should replace it by an invariant that depends on the type A

Intuition:

The more complex the type, the stronger its invariant,
the smaller the set of terms that fulfill this invariant

Reducibility candidates [Girard 1971]

To prove that

$$\Gamma \vdash t : A \quad \Rightarrow \quad t \text{ is SN}\,,$$

the induction hypothesis "t is SN" is too weak.

$\Rightarrow$ Should replace it by an invariant that depends on the type A

Intuition:

The more complex the type, the stronger its invariant,
the smaller the set of terms that fulfill this invariant

Invariants are represented by suitable sets of terms:

Reducibility candidates [Girard 1971]

To prove that

$$\Gamma \vdash t : A \quad \Rightarrow \quad t \text{ is SN}\,,$$

the induction hypothesis "t is SN" is too weak.

$\Rightarrow$ Should replace it by an invariant that depends on the type A

Intuition:

The more complex the type, the stronger its invariant,
the smaller the set of terms that fulfill this invariant

Invariants are represented by suitable sets of terms:

- Reducibility candidates [Girard]

Reducibility candidates [Girard 1971]

To prove that

$$\Gamma \vdash t : A \quad \Rightarrow \quad t \text{ is SN},$$

the induction hypothesis "t is SN" is too weak.

$\Rightarrow$ Should replace it by an invariant that depends on the type A

Intuition:

The more complex the type, the stronger its invariant,
the smaller the set of terms that fulfill this invariant

Invariants are represented by suitable sets of terms:

- Reducibility candidates [Girard], or
- Saturated sets [Tait]

Outline of the proof

1. Define a suitable notion of reducibility candidate
   = the sets of λ-terms that will interpret/represent types

   (Here, we use Tait's saturated sets)

Outline of the proof

1. Define a suitable notion of reducibility candidate
   $=$ the sets of λ-terms that will interpret/represent types

   (Here, we use Tait's saturated sets)

2. Ensure that the notion of candidate captures the property of strong normalisation (which we want to prove)

   Each candidate should only contain strongly normalisable λ-terms as elements

Outline of the proof

1. Define a suitable notion of reducibility candidate
   $=$ the sets of λ-terms that will interpret/represent types

   (Here, we use Tait's saturated sets)

2. Ensure that the notion of candidate captures the property of strong normalisation (which we want to prove)

   Each candidate should only contain strongly normalisable λ-terms as elements

3. Associate to each type A a reducibility candidate $[\![A]\!]$

   Type constructors '$\rightarrow$' and '$\forall$' have to be reflected at the level of candidates

Outline of the proof

1. Define a suitable notion of reducibility candidate
   $=$ the sets of λ-terms that will interpret/represent types

   (Here, we use Tait's saturated sets)

2. Ensure that the notion of candidate captures the property of strong normalisation (which we want to prove)

   Each candidate should only contain strongly normalisable λ-terms as elements

3. Associate to each type A a reducibility candidate $[\![A]\!]$

   Type constructors '$\rightarrow$' and '$\forall$' have to be reflected at the level of candidates

4. Check (by induction) that $\Gamma \vdash t : A$ implies $t \in [\![A]\!]$

   This is actually a little bit more complex, since we must take care of the typing context

Outline of the proof

1. Define a suitable notion of reducibility candidate
   $=$ the sets of λ-terms that will interpret/represent types

   (Here, we use Tait's saturated sets)

2. Ensure that the notion of candidate captures the property of strong normalisation (which we want to prove)

   Each candidate should only contain strongly normalisable λ-terms as elements

3. Associate to each type A a reducibility candidate $[\![A]\!]$

   Type constructors '$\to$' and '$\forall$' have to be reflected at the level of candidates

4. Check (by induction) that $\Gamma \vdash t : A$ implies $t \in [\![A]\!]$

   This is actually a little bit more complex, since we must take care of the typing context

5. Conclude that any well-typed term t is SN by step 2.

Preliminaries (1/2)

Preliminaries (1/2)

- **Notations:**

Λ	$\equiv$	set of all untyped λ-terms (open & closed)
SN	$\equiv$	set of all strongly normalisable untyped λ-terms
Var	$\equiv$	set of all (term) variables
TVar	$\equiv$	set of all type variables

Preliminaries (1/2)

- **Notations:**

Λ	$\equiv$	set of all untyped λ-terms (open & closed)
SN	$\equiv$	set of all strongly normalisable untyped λ-terms
Var	$\equiv$	set of all (term) variables
TVar	$\equiv$	set of all type variables

- A reduct of a term t is a term t' such that $t \succ t'$ (one step)

  The number of reducts of a given term is finite and bounded by the number of redexes

Preliminaries (1/2)

- **Notations:**

Λ	$\equiv$	set of all untyped λ-terms (open & closed)
SN	$\equiv$	set of all strongly normalisable untyped λ-terms
Var	$\equiv$	set of all (term) variables
TVar	$\equiv$	set of all type variables

- A reduct of a term t is a term t' such that $t \succ t'$ (one step)

  The number of reducts of a given term is finite and bounded by the number of redexes

- A finite reduction sequence of a term t is a finite sequence $(t_i)_{i \in [0..n]}$ such that $t = t_0 \succ t_1 \succ \cdots \succ t_{n-1} \succ t_n$

  Infinite reduction sequences are defined similarly, by replacing $[0..n]$ by $\mathbb{N}$

Preliminaries (1/2)

- **Notations:**

Λ	$\equiv$	set of all untyped λ-terms (open & closed)
SN	$\equiv$	set of all strongly normalisable untyped λ-terms
Var	$\equiv$	set of all (term) variables
TVar	$\equiv$	set of all type variables

- A reduct of a term t is a term t' such that $t \succ t'$ (one step)

  The number of reducts of a given term is finite and bounded by the number of redexes

- A finite reduction sequence of a term t is a finite sequence $(t_i)_{i \in [0..n]}$ such that $t = t_0 \succ t_1 \succ \cdots \succ t_{n-1} \succ t_n$

  Infinite reduction sequences are defined similarly, by replacing $[0..n]$ by $\mathbb{N}$

- Finite reduction sequences of a term t form a tree, called the reduction tree of t

Preliminaries (2/2)

Preliminaries (2/2)

Definition (Strongly normalisable terms)

A term t is strongly normalisable if all the reduction sequences starting from t are finite

Preliminaries (2/2)

Definition (Strongly normalisable terms)

A term t is strongly normalisable if all the reduction sequences starting from t are finite

Proposition

The following assertions are equivalent:

1. t is strongly normalisable
2. All the reducts of t are strongly normalisable
3. The reduction tree of t is finite

Saturated sets [Tait]

Definition (Saturated set)

A set $S \subset \Lambda$ is saturated if:

$$
\begin{array}{lll}
\text{(SAT1)} & S \subset \mathsf{SN} & \\
\text{(SAT2)} & x \in \mathsf{Var}, \quad \vec{v} \in \mathsf{list}(\mathsf{SN}) & \Rightarrow \quad x\vec{v} \in S \\
\text{(SAT3)} & t\{x := u\}\vec{v} \in S, \quad u \in \mathsf{SN} & \Rightarrow \quad (\lambda x \,.\, t)u\vec{v} \in S
\end{array}
$$

Saturated sets [Tait]

Definition (Saturated set)

A set $S \subset \Lambda$ is saturated if:

$$\begin{array}{lll} \text{(SAT1)} & S \subset \mathsf{SN} & \\ \text{(SAT2)} & x \in \mathsf{Var}, \quad \vec{v} \in \mathsf{list}(\mathsf{SN}) \quad \Rightarrow \quad x\vec{v} \in S & \\ \text{(SAT3)} & t\{x := u\}\vec{v} \in S, \quad u \in \mathsf{SN} \quad \Rightarrow \quad (\lambda x\,.\,t)u\vec{v} \in S & \end{array}$$

- (SAT1) expresses the property we want to prove

Saturated sets [Tait]

Definition (Saturated set)

A set $S \subset \Lambda$ is saturated if:

$$\begin{array}{lll}
\text{(SAT1)} & S \subset \mathsf{SN} & \\
\text{(SAT2)} & x \in \mathsf{Var}, \quad \vec{v} \in \mathsf{list}(\mathsf{SN}) & \Rightarrow \quad x\vec{v} \in S \\
\text{(SAT3)} & t\{x := u\}\vec{v} \in S, \quad u \in \mathsf{SN} & \Rightarrow \quad (\lambda x \,.\, t)u\vec{v} \in S
\end{array}$$

- (SAT1) expresses the property we want to prove
- Saturated sets contain all the variables (SAT2)
  Extra-arguments $\vec{v} \in \mathsf{list}(\mathsf{SN})$ are here for technical reasons

Saturated sets [Tait]

Definition (Saturated set)

A set $S \subset \Lambda$ is saturated if:

(SAT1)	$S \subset \mathrm{SN}$
(SAT2)	$x \in \mathrm{Var}, \quad \vec{v} \in \mathrm{list}(\mathrm{SN}) \quad \Rightarrow \quad x\vec{v} \in S$
(SAT3)	$t\{x := u\}\vec{v} \in S, \quad u \in \mathrm{SN} \quad \Rightarrow \quad (\lambda x \,.\, t)u\vec{v} \in S$

- (SAT1) expresses the property we want to prove
- Saturated sets contain all the variables (SAT2)
  Extra-arguments $\vec{v} \in \mathrm{list}(\mathrm{SN})$ are here for technical reasons
- Saturated sets are closed under head β-expansion (SAT3)
  Notice the condition $u \in \mathrm{SN}$ to avoid a clash with (SAT1) for K-redexes

Saturated sets [Tait]

Definition (Saturated set)

A set $S \subset \Lambda$ is saturated if:

(SAT1)	$S \subset \mathrm{SN}$	
(SAT2)	$x \in \mathrm{Var}, \quad \vec{v} \in \mathrm{list}(\mathrm{SN})$	$\Rightarrow \quad x\vec{v} \in S$
(SAT3)	$t\{x := u\}\vec{v} \in S, \quad u \in \mathrm{SN}$	$\Rightarrow \quad (\lambda x \,.\, t)u\vec{v} \in S$

- (SAT1) expresses the property we want to prove
- Saturated sets contain all the variables (SAT2)
  Extra-arguments $\vec{v} \in \mathrm{list}(\mathrm{SN})$ are here for technical reasons
- Saturated sets are closed under head β-expansion (SAT3)
  Notice the condition $u \in \mathrm{SN}$ to avoid a clash with (SAT1) for K-redexes
- The set of all saturated sets is written **SAT** $[\subset \mathfrak{P}(\mathrm{SN}) \subset \mathfrak{P}(\Lambda)]$

Properties of saturated sets

Proposition (Lattice structure)

Properties of saturated sets

Proposition (Lattice structure)

1. SN is a saturated set

Properties of saturated sets

Proposition (Lattice structure)

1. SN is a saturated set
2. **SAT** is closed under arbitrary non-empty intersections/unions:

$$I \neq \varnothing, \quad (S_i)_{i \in I} \in \mathbf{SAT}^I \quad \Rightarrow \quad \left(\bigcap_{i \in I} S_i\right), \left(\bigcup_{i \in I} S_i\right) \in \mathbf{SAT}$$

Properties of saturated sets

Proposition (Lattice structure)

1. SN is a saturated set
2. **SAT** is closed under arbitrary non-empty intersections/unions:

$$I \neq \varnothing, \quad (S_i)_{i \in I} \in \mathbf{SAT}^I \quad \Rightarrow \quad \left(\bigcap_{i \in I} S_i\right), \left(\bigcup_{i \in I} S_i\right) \in \mathbf{SAT}$$

($\mathbf{SAT}$, $\subset$) is a complete distributive lattice, with

$\top = \mathrm{SN}$ and $\bot = \{t \in \mathrm{SN} \mid t \succ^* x u_1 \cdots u_n\}$ (Neutral terms)

Properties of saturated sets

Proposition (Lattice structure)

1. SN is a saturated set
2. **SAT** is closed under arbitrary non-empty intersections/unions:

$$I \neq \varnothing, \quad (S_i)_{i \in I} \in \mathbf{SAT}^I \quad \Rightarrow \quad \Big(\bigcap_{i \in I} S_i\Big), \Big(\bigcup_{i \in I} S_i\Big) \in \mathbf{SAT}$$

$(\mathbf{SAT}, \subset)$ is a complete distributive lattice, with
$\top = \mathrm{SN}$ and $\bot = \{t \in \mathrm{SN} \mid t \succ^* x u_1 \cdots u_n\}$ (Neutral terms)

Realisability arrow: For all $S, T \subset \Lambda$ we set

$$S \to T \quad := \quad \{t \in \Lambda \mid \forall u \in S \quad tu \in T\}$$

Properties of saturated sets

Proposition (Lattice structure)

1. SN is a saturated set
2. **SAT** is closed under arbitrary non-empty intersections/unions:

$$I \neq \varnothing, \quad (S_i)_{i \in I} \in \mathbf{SAT}^I \quad \Rightarrow \quad \Big(\bigcap_{i \in I} S_i\Big), \Big(\bigcup_{i \in I} S_i\Big) \in \mathbf{SAT}$$

$(\mathbf{SAT}, \subset)$ is a complete distributive lattice, with
$\top = \mathrm{SN}$ and $\bot = \{t \in \mathrm{SN} \mid t \succ^* x u_1 \cdots u_n\}$ (Neutral terms)

Realisability arrow: For all $S, T \subset \Lambda$ we set

$$S \to T \quad := \quad \{t \in \Lambda \mid \forall u \in S \quad tu \in T\}$$

Proposition (Closure under realisability arrow)

If $S, T \in \mathbf{SAT}$, then $(S \to T) \in \mathbf{SAT}$

Interpreting types (1/2)

Principle: Interpret syntactic types by saturated sets

Interpreting types (1/2)

Principle: Interpret syntactic types by saturated sets

- Type arrow $A \rightarrow B$ is interpreted by $S \rightarrow T$ (realisability arrow)

Interpreting types (1/2)

Principle: Interpret syntactic types by saturated sets

- Type arrow $A \to B$ is interpreted by $S \to T$ (realisability arrow)
- Type quantification $\forall\alpha$ .. is interpreted by the intersection $\bigcap_{S \in \mathbf{SAT}} \cdots$

Interpreting types (1/2)

Principle: Interpret syntactic types by saturated sets

- Type arrow $A \to B$ is interpreted by $S \to T$ (realisability arrow)
- Type quantification $\forall\alpha$ .. is interpreted by the intersection $\bigcap_{S \in \mathsf{SAT}} \cdots$

Remark: this intersection is impredicative since S ranges over all saturated sets

Interpreting types (1/2)

Principle: Interpret syntactic types by saturated sets

- Type arrow $A \rightarrow B$ is interpreted by $S \rightarrow T$ (realisability arrow)
- Type quantification $\forall\alpha$ .. is interpreted by the intersection $\bigcap_{S \in \mathsf{SAT}} \cdots$

Remark: this intersection is impredicative since S ranges over all saturated sets

Example: $\forall\alpha\ (\alpha \rightarrow \alpha)$ should be interpreted by $\bigcap_{S \in \mathsf{SAT}} (S \rightarrow S)$

Interpreting types (1/2)

Principle: Interpret syntactic types by saturated sets

- Type arrow $A \to B$ is interpreted by $S \to T$ (realisability arrow)
- Type quantification $\forall\alpha$ .. is interpreted by the intersection $\bigcap_{S \in \mathsf{SAT}} \cdots$

Remark: this intersection is impredicative since S ranges over all saturated sets

Example: $\forall\alpha\ (\alpha \to \alpha)$ should be interpreted by $\bigcap_{S \in \mathsf{SAT}} (S \to S)$

To interpret type variables, use type valations:

Interpreting types (1/2)

Principle: Interpret syntactic types by saturated sets

- Type arrow $A \to B$ is interpreted by $S \to T$ (realisability arrow)
- Type quantification $\forall\alpha$ .. is interpreted by the intersection $\bigcap_{S \in \mathbf{SAT}} \cdots$

Remark: this intersection is impredicative since S ranges over all saturated sets

Example: $\forall\alpha\ (\alpha \to \alpha)$ should be interpreted by $\bigcap_{S \in \mathbf{SAT}} (S \to S)$

To interpret type variables, use type valations:

Definition (Type valuations)

A type valuation is a function $\rho : \mathrm{TVar} \to \mathbf{SAT}$

The set of type valuations is written TVal $(= \mathrm{TVar} \to \mathbf{SAT})$

Interpreting types (2/2)

By induction on A, we define a function $[\![A]\!] : \mathsf{TVal} \to \mathbf{SAT}$

Interpreting types (2/2)

By induction on A, we define a function $\llbracket A \rrbracket : \mathrm{TVal} \to \mathbf{SAT}$

$$\llbracket A \to B \rrbracket_\rho = \llbracket A \rrbracket_\rho \to \llbracket B \rrbracket_\rho \qquad \llbracket \alpha \rrbracket_\rho = \rho(\alpha)$$

$$\llbracket \forall\alpha\ B \rrbracket_\rho = \bigcap_{S \in \mathbf{SAT}} \llbracket B \rrbracket_{\rho;\alpha\leftarrow S}$$

Note: $(\rho;\alpha \leftarrow S)$ is defined by $\begin{cases} (\rho;\alpha{\leftarrow}S)(\alpha) = S \\ (\rho;\alpha{\leftarrow}S)(\beta) = \rho(\beta) & \text{for all } \beta \neq \alpha \end{cases}$

Interpreting types (2/2)

By induction on A, we define a function $[\![A]\!] : \mathsf{TVal} \to \mathbf{SAT}$

$$[\![A \to B]\!]_\rho = [\![A]\!]_\rho \to [\![B]\!]_\rho \qquad [\![\alpha]\!]_\rho = \rho(\alpha)$$

$$[\![\forall \alpha\ B]\!]_\rho = \bigcap_{S \in \mathbf{SAT}} [\![B]\!]_{\rho;\alpha \leftarrow S}$$

Note: $(\rho; \alpha \leftarrow S)$ is defined by $\begin{cases} (\rho; \alpha \leftarrow S)(\alpha) = S \\ (\rho; \alpha \leftarrow S)(\beta) = \rho(\beta) & \text{for all } \beta \neq \alpha \end{cases}$

Problem: The implication

$$\Gamma \vdash t : A \quad \Rightarrow \quad t \in [\![A]\!]_\rho$$

cannot be proved directly. (One has to take care of the context)

Interpreting types (2/2)

By induction on A, we define a function $[\![A]\!] : \text{TVal} \to \mathbf{SAT}$

$$[\![A \to B]\!]_\rho \;=\; [\![A]\!]_\rho \to [\![B]\!]_\rho \qquad\qquad [\![\alpha]\!]_\rho \;=\; \rho(\alpha)$$

$$[\![\forall\alpha\ B]\!]_\rho \;=\; \bigcap_{S \in \mathbf{SAT}} [\![B]\!]_{\rho;\alpha\leftarrow S}$$

Note: $(\rho; \alpha \leftarrow S)$ is defined by $\begin{cases}(\rho;\alpha\leftarrow S)(\alpha) = S \\ (\rho;\alpha\leftarrow S)(\beta) = \rho(\beta) & \text{for all } \beta \neq \alpha\end{cases}$

Problem: The implication

$$\Gamma \vdash t : A \quad\Rightarrow\quad t \in [\![A]\!]_\rho$$

cannot be proved directly. (One has to take care of the context)

$\Rightarrow$ Strengthen induction hypothesis using substitutions

Substitutions

Substitutions

Definition (Substitutions)

A substitution is a finite list $\sigma = [x_1 := u_1; \ldots; x_n := u_n]$ where $x_i \neq x_j$ (for $i \neq j$) and $u_i \in \Lambda$

Substitutions

Definition (Substitutions)

A substitution is a finite list $\sigma = [x_1 := u_1 ; \ldots ; x_n := u_n]$ where $x_i \neq x_j$ (for $i \neq j$) and $u_i \in \Lambda$

Application of a substitution σ to a term t is written $t[\sigma]$

Exercise: Define it formally

Substitutions

Definition (Substitutions)

A substitution is a finite list $\sigma = [x_1 := u_1; \ldots; x_n := u_n]$ where $x_i \neq x_j$ (for $i \neq j$) and $u_i \in \Lambda$

Application of a substitution σ to a term t is written $t[\sigma]$

Exercise: Define it formally

Definition (Interpretation of contexts)

For all $\Gamma = x_1 : A_1; \ldots; x_n : A_n$ and $\rho \in \mathrm{TVal}$ set:

$$[\![\Gamma]\!]_\rho = \{\sigma = [x_1 := u_1; \ldots; x_n := u_n]; \quad u_i \in [\![A_i]\!]_\rho \ \ (i = 1..n)\}$$

Substitutions $\sigma \in [\![\Gamma]\!]_\rho$ are said to be adapted to the context Γ (in the type valuation ρ)

The strong normalisation invariant

The strong normalisation invariant

Lemma (Strong normalisation invariant)

If $\Gamma \vdash t : A$ in Curry-style system F, then

$$\forall \rho \in \mathsf{TVal} \qquad \forall \sigma \in [\![\Gamma]\!]_\rho \qquad t[\sigma] \in [\![A]\!]_\rho$$

The strong normalisation invariant

Lemma (Strong normalisation invariant)

If $\Gamma \vdash t : A$ in Curry-style system F, then

$$\forall \rho \in \mathsf{TVal} \qquad \forall \sigma \in [\![\Gamma]\!]_\rho \qquad t[\sigma] \in [\![A]\!]_\rho$$

Proof. By induction on the derivation of $\Gamma \vdash t : A$.

Exercise: Write down the 5 cases completely

The strong normalisation invariant

Lemma (Strong normalisation invariant)

If $\Gamma \vdash t : A$ in Curry-style system F, then

$$\forall \rho \in \mathrm{TVal} \qquad \forall \sigma \in [\![\Gamma]\!]_\rho \qquad t[\sigma] \in [\![A]\!]_\rho$$

Proof. By induction on the derivation of $\Gamma \vdash t : A$.

Exercise: Write down the 5 cases completely

Theorem (Strong normalisation)

The typable terms of F-Curry are strongly normalisable

The strong normalisation invariant

Lemma (Strong normalisation invariant)

If $\Gamma \vdash t : A$ in Curry-style system F, then

$$\forall \rho \in \mathsf{TVal} \qquad \forall \sigma \in [\![\Gamma]\!]_\rho \qquad t[\sigma] \in [\![A]\!]_\rho$$

Proof. By induction on the derivation of $\Gamma \vdash t : A$.

Exercise: Write down the 5 cases completely

Theorem (Strong normalisation)

The typable terms of F-Curry are strongly normalisable

Proof. Assume $x_1 : A_1; \ldots; x_n : A_n \vdash t : B$

The strong normalisation invariant

Lemma (Strong normalisation invariant)

If $\Gamma \vdash t : A$ in Curry-style system F, then

$$\forall \rho \in \mathsf{TVal} \qquad \forall \sigma \in [\![\Gamma]\!]_\rho \qquad t[\sigma] \in [\![A]\!]_\rho$$

Proof. By induction on the derivation of $\Gamma \vdash t : A$.

Exercise: Write down the 5 cases completely

Theorem (Strong normalisation)

The typable terms of F-Curry are strongly normalisable

Proof. Assume $x_1 : A_1; \ldots; x_n : A_n \vdash t : B$

Consider an arbitrary type valuation ρ (for instance: $\rho(\alpha) = \mathsf{SN}$ for all α)

The strong normalisation invariant

Lemma (Strong normalisation invariant)

If $\Gamma \vdash t : A$ in Curry-style system F, then

$$\forall \rho \in \text{TVal} \qquad \forall \sigma \in [\![\Gamma]\!]_\rho \qquad t[\sigma] \in [\![A]\!]_\rho$$

Proof. By induction on the derivation of $\Gamma \vdash t : A$.

Exercise: Write down the 5 cases completely

Theorem (Strong normalisation)

The typable terms of F-Curry are strongly normalisable

Proof. Assume $x_1 : A_1; \ldots; x_n : A_n \vdash t : B$

Consider an arbitrary type valuation ρ (for instance: $\rho(\alpha) = \text{SN}$ for all α)

We have: $x_1 \in [\![A_1]\!]_\rho$, $x_2 \in [\![A_2]\!]_\rho$, ..., $x_n \in [\![A_n]\!]_\rho$ (SAT2)

The strong normalisation invariant

Lemma (Strong normalisation invariant)

If $\Gamma \vdash t : A$ in Curry-style system F, then

$$\forall \rho \in \mathrm{TVal} \qquad \forall \sigma \in [\![\Gamma]\!]_\rho \qquad t[\sigma] \in [\![A]\!]_\rho$$

Proof. By induction on the derivation of $\Gamma \vdash t : A$.

Exercise: Write down the 5 cases completely

Theorem (Strong normalisation)

The typable terms of F-Curry are strongly normalisable

Proof. Assume $x_1 : A_1; \ldots; x_n : A_n \vdash t : B$

Consider an arbitrary type valuation ρ (for instance: $\rho(\alpha) = \mathrm{SN}$ for all α)

We have: $x_1 \in [\![A_1]\!]_\rho$, $x_2 \in [\![A_2]\!]_\rho$, ..., $x_n \in [\![A_n]\!]_\rho$ (SAT2), hence:

$$\sigma = [x_1 := x_1; \ldots; x_n := x_n] \in [\![x_1 : A_1; \ldots; x_n : A_n]\!]_\rho$$

From the lemma we get $t = t[\sigma] \in [\![B]\!]_\rho$, hence $t \in \mathrm{SN}$ (SAT1)

The strong normalisation invariant

Lemma (Strong normalisation invariant)

If $\Gamma \vdash t : A$ in Curry-style system F, then

$$\forall \rho \in \mathsf{TVal} \qquad \forall \sigma \in [\![\Gamma]\!]_\rho \qquad t[\sigma] \in [\![A]\!]_\rho$$

Proof. By induction on the derivation of $\Gamma \vdash t : A$.

Exercise: Write down the 5 cases completely

Theorem (Strong normalisation)

The typable terms of F-Curry are strongly normalisable

Corollary (Church-style SN)

The typable terms of F-Church are strongly normalisable

A remark on impredicativity

In the SN proof, interpretation of $\forall$ relies on the property:

If $(S_i)_{i\in I}$ $(I \neq \varnothing)$ is a family of saturated sets,
then $\bigcap_{i\in I} S_i$ is a saturated set

in the special case where $I = \mathbf{SAT}$ (impredicative intersection)

A remark on impredicativity

In the SN proof, interpretation of $\forall$ relies on the property:

If $(S_i)_{i \in I}$ $(I \neq \varnothing)$ is a family of saturated sets,
then $\bigcap_{i \in I} S_i$ is a saturated set

in the special case where $I = \mathbf{SAT}$ (impredicative intersection)

- In 'classical' mathematics, this construction is legal

A remark on impredicativity

In the SN proof, interpretation of $\forall$ relies on the property:

If $(S_i)_{i\in I}$ $(I \neq \varnothing)$ is a family of saturated sets,
then $\bigcap_{i\in I} S_i$ is a saturated set

in the special case where $I = \mathbf{SAT}$ (impredicative intersection)

- In 'classical' mathematics, this construction is legal

  $\Rightarrow$ Standard set theories (Z, ZF, ZFC) are impredicative

A remark on impredicativity

In the SN proof, interpretation of $\forall$ relies on the property:

If $(S_i)_{i\in I}$ $(I \neq \varnothing)$ is a family of saturated sets,
then $\bigcap_{i\in I} S_i$ is a saturated set

in the special case where $I = \mathbf{SAT}$ (impredicative intersection)

- In 'classical' mathematics, this construction is legal

  $\Rightarrow$ Standard set theories (Z, ZF, ZFC) are impredicative

- In (Bishop, Martin-Löf's style) constructive mathematics, this principle is rejected, mainly for philosophical reasons:
  - No convincing 'constructive' explanation
  - Suspicion about (this kind of) cyclicity

Impredicativity: An example (1/2)

Assume E is a vector space, S a set of vectors.
How to define the sub-vector space $\overline{S} \subset E$ generated by S in E ?

Impredicativity: An example (1/2)

Assume E is a vector space, S a set of vectors.
How to define the sub-vector space $\overline{S} \subset E$ generated by S in E ?

Standard 'abstract' method:

Impredicativity: An example (1/2)

Assume E is a vector space, S a set of vectors.
How to define the sub-vector space $\overline{S} \subset E$ generated by S in E ?

Standard 'abstract' method:

1. Consider the set: $\mathfrak{S} = \{F; \quad F \text{ is a sub-vector space of } E \text{ and } F \supset S\}$

Impredicativity: An example (1/2)

Assume E is a vector space, S a set of vectors.
How to define the sub-vector space $\overline{S} \subset E$ generated by S in E ?

Standard 'abstract' method:

1. Consider the set: $\mathfrak{S} = \{F; \quad F \text{ is a sub-vector space of } E \text{ and } F \supset S\}$
2. Fact: $\mathfrak{S}$ is non empty, since $E \in \mathfrak{S}$

Impredicativity: An example (1/2)

Assume E is a vector space, S a set of vectors.
How to define the sub-vector space $\overline{S} \subset E$ generated by S in E ?

Standard 'abstract' method:

1. Consider the set: $\mathfrak{S} = \{F; \quad F \text{ is a sub-vector space of } E \text{ and } F \supset S\}$
2. Fact: $\mathfrak{S}$ is non empty, since $E \in \mathfrak{S}$
3. Take: $\overline{S} = \bigcap_{F \in \mathfrak{S}} F$

Impredicativity: An example (1/2)

Assume E is a vector space, S a set of vectors.
How to define the sub-vector space $\overline{S} \subset E$ generated by S in E ?

Standard 'abstract' method:

1. Consider the set: $\mathfrak{S} = \{F; \quad F \text{ is a sub-vector space of } E \text{ and } F \supset S\}$
2. Fact: $\mathfrak{S}$ is non empty, since $E \in \mathfrak{S}$
3. Take: $\overline{S} = \bigcap_{F \in \mathfrak{S}} F$
4. By definition, S is included in all the sub-spaces of E containing S

Impredicativity: An example (1/2)

Assume E is a vector space, S a set of vectors.
How to define the sub-vector space $\overline{S} \subset E$ generated by S in E ?

Standard 'abstract' method:

1. Consider the set: $\mathfrak{S} = \{F; \quad F \text{ is a sub-vector space of } E \text{ and } F \supset S\}$
2. Fact: $\mathfrak{S}$ is non empty, since $E \in \mathfrak{S}$
3. Take: $\overline{S} = \bigcap_{F \in \mathfrak{S}} F$
4. By definition, S is included in all the sub-spaces of E containing S
5. But $\overline{S}$ is itself a sub-vector space of E containing S (so that $\overline{S} \in \mathfrak{S}$)

Impredicativity: An example (1/2)

Assume E is a vector space, S a set of vectors.
How to define the sub-vector space $\overline{S} \subset E$ generated by S in E ?

Standard 'abstract' method:

1. Consider the set: $\mathfrak{S} = \{F; \quad F \text{ is a sub-vector space of } E \text{ and } F \supset S\}$
2. Fact: $\mathfrak{S}$ is non empty, since $E \in \mathfrak{S}$
3. Take: $\overline{S} = \bigcap_{F \in \mathfrak{S}} F$
4. By definition, S is included in all the sub-spaces of E containing S
5. But $\overline{S}$ is itself a sub-vector space of E containing S (so that $\overline{S} \in \mathfrak{S}$)
6. So that $\overline{S}$ is actually the smallest of all such spaces

Impredicativity: An example (1/2)

Assume E is a vector space, S a set of vectors.
How to define the sub-vector space $\overline{S} \subset E$ generated by S in E ?

Standard 'abstract' method:

1. Consider the set: $\mathfrak{S} = \{F; \quad F \text{ is a sub-vector space of } E \text{ and } F \supset S\}$
2. Fact: $\mathfrak{S}$ is non empty, since $E \in \mathfrak{S}$
3. Take: $\overline{S} = \bigcap_{F \in \mathfrak{S}} F$
4. By definition, S is included in all the sub-spaces of E containing S
5. But $\overline{S}$ is itself a sub-vector space of E containing S (so that $\overline{S} \in \mathfrak{S}$)
6. So that $\overline{S}$ is actually the smallest of all such spaces

This definition is impredicative (step 3) (but legal in 'classical' mathematics)

Impredicativity: An example (1/2)

Assume E is a vector space, S a set of vectors.
How to define the sub-vector space $\overline{S} \subset E$ generated by S in E ?

Standard 'abstract' method:

1. Consider the set: $\mathfrak{S} = \{F; \quad F \text{ is a sub-vector space of } E \text{ and } F \supset S\}$
2. Fact: $\mathfrak{S}$ is non empty, since $E \in \mathfrak{S}$
3. Take: $\overline{S} = \bigcap_{F \in \mathfrak{S}} F$
4. By definition, S is included in all the sub-spaces of E containing S
5. But $\overline{S}$ is itself a sub-vector space of E containing S (so that $\overline{S} \in \mathfrak{S}$)
6. So that $\overline{S}$ is actually the smallest of all such spaces

This definition is impredicative (step 3) (but legal in 'classical' mathematics)

The set $\overline{S}$ is defined *from* $\mathfrak{S}$, $\underbrace{\text{that already contains } \overline{S} \textit{ as an element}}_{\text{discovered a fortiori}}$

Impredicativity: An example (2/2)

But there are other ways of defining $\overline{S}$...

Impredicativity: An example (2/2)

But there are other ways of defining $\overline{S}$...

- **Standard 'concrete' definition, by linear combinations:**

Impredicativity: An example (2/2)

But there are other ways of defining $\overline{S}$...

- **Standard 'concrete' definition, by linear combinations:**

  Let $\overline{S}$ be the set of all vectors of the form $v = \alpha_1 \cdot v_1 + \cdots + \alpha_n \cdot v_n$

  where (v_i) ranges over all the finite families of elements of S,
  and (α_i) ranges over all the finite families of scalars

Impredicativity: An example (2/2)

But there are other ways of defining $\overline{S}$...

- **Standard 'concrete' definition, by linear combinations:**

  Let $\overline{S}$ be the set of all vectors of the form $v = \alpha_1 \cdot v_1 + \cdots + \alpha_n \cdot v_n$

  where (v_i) ranges over all the finite families of elements of S,
  and (α_i) ranges over all the finite families of scalars

- **Inductive definition:**

Impredicativity: An example (2/2)

But there are other ways of defining $\overline{S}$...

- **Standard 'concrete' definition, by linear combinations:**

  Let $\overline{S}$ be the set of all vectors of the form $v = \alpha_1 \cdot v_1 + \cdots + \alpha_n \cdot v_n$

  where (v_i) ranges over all the finite families of elements of S,
  and (α_i) ranges over all the finite families of scalars

- **Inductive definition:**

  Let $\overline{S}$ be the set inductively defined by:

  1. $\vec{0} \in \overline{S}$,
  2. If $v \in S$, then $v \in \overline{S}$,
  3. If $v \in \overline{S}$ and α is a scalar, then $\alpha \cdot v \in \overline{S}$
  4. If $v_1 \in \overline{S}$ and $v_2 \in \overline{S}$, then $v_1 + v_2 \in \overline{S}$.

Impredicativity: An example (2/2)

But there are other ways of defining $\overline{S}$...

- **Standard 'concrete' definition, by linear combinations:**

  Let $\overline{S}$ be the set of all vectors of the form $\quad v = \alpha_1 \cdot v_1 + \cdots + \alpha_n \cdot v_n$

  where $\quad (v_i)$ ranges over all the finite families of elements of S,
  and $\quad (\alpha_i)$ ranges over all the finite families of scalars

- **Inductive definition:**

  Let $\overline{S}$ be the set inductively defined by:

  1. $\vec{0} \in \overline{S}$,
  2. If $v \in S$, then $v \in \overline{S}$,
  3. If $v \in \overline{S}$ and α is a scalar, then $\alpha \cdot v \in \overline{S}$
  4. If $v_1 \in \overline{S}$ and $v_2 \in \overline{S}$, then $v_1 + v_2 \in \overline{S}$.

$\Rightarrow$ Both definitions are predicative (and give the same object)