Type Theory and Coq

Radboud University Nijmegen, The Netherlands

Lecture 3

The Church-Rosser Property and Principal Types

Today's lecture

What do we want to prove about type systems? So: what about the meta theory of type theory

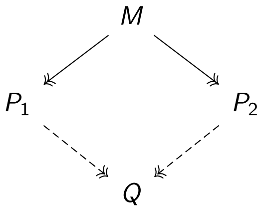
Today's lecture

What do we want to prove about type systems? So: what about the meta theory of type theory

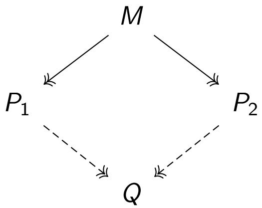
- Church-Rosser (confluence) of reduction
- Type inference (inferring principal types)

More properties are of interest, such as (strong) normalization, but that is not for today

Part 1: Church-Rosser property, CR

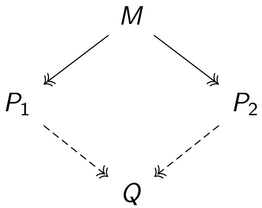


Part 1: Church-Rosser property, CR



CHURCH-ROSSER THEOREM for β-reduction, CR_β.
If $M \twoheadrightarrow_\beta P_1$ and $M \twoheadrightarrow_\beta P_2$, then $\exists Q(P_1 \twoheadrightarrow_\beta Q \wedge P_2 \twoheadrightarrow_\beta Q)$

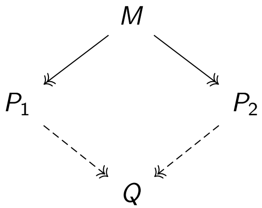
Part 1: Church-Rosser property, CR



CHURCH-ROSSER THEOREM for β-reduction, CR_β.
If $M \twoheadrightarrow_\beta P_1$ and $M \twoheadrightarrow_\beta P_2$, then $\exists Q(P_1 \twoheadrightarrow_\beta Q \wedge P_2 \twoheadrightarrow_\beta Q)$

NB. $M \twoheadrightarrow P$ denotes the reflexive transitive closure of $M \to P$, that is:
$M \twoheadrightarrow P$ iff there is a multi-step (0 or more) reduction from M to P.

Part 1: Church-Rosser property, CR



Church-Rosser Theorem for β-reduction, CR_β.
If $M \twoheadrightarrow_\beta P_1$ and $M \twoheadrightarrow_\beta P_2$, then $\exists Q(P_1 \twoheadrightarrow_\beta Q \wedge P_2 \twoheadrightarrow_\beta Q)$

NB. $M \twoheadrightarrow P$ denotes the reflexive transitive closure of $M \rightarrow P$, that is:
$M \twoheadrightarrow P$ iff there is a multi-step (0 or more) reduction from M to P.

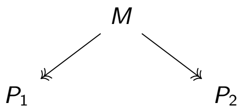
We will prove the Church-Rosser Theorem for β-reduction of the untyped λ-calculus in this lecture.

Church-Rosser (for β) example

$$(\lambda x. y\, x\, x)(\mathbf{I}\,\mathbf{I})$$

Corollaries of the Church-Rosser property

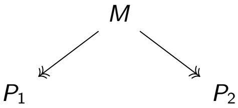
THEOREM CR($\rightarrow_R$) implies UN($\rightarrow_R$) (Uniqueness of Normal forms)



If P_1 and P_2 are in normal form, then $P_1 = P_2$, due to CR.

Corollaries of the Church-Rosser property

THEOREM $\mathsf{CR}(\rightarrow_R)$ implies $\mathsf{UN}(\rightarrow_R)$ (Uniqueness of Normal forms)

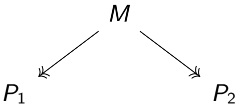


If P_1 and P_2 are in normal form, then $P_1 = P_2$, due to CR.

THEOREM $\mathsf{CR}(\rightarrow_R) + \mathsf{SN}(\rightarrow_R)$ implies $=_R$ is decidable.

Corollaries of the Church-Rosser property

THEOREM $\mathsf{CR}(\rightarrow_R)$ implies $\mathsf{UN}(\rightarrow_R)$ (Uniqueness of Normal forms)



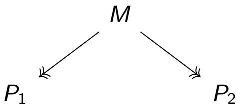
If P_1 and P_2 are in normal form, then $P_1 = P_2$, due to CR.

THEOREM $\mathsf{CR}(\rightarrow_R) + \mathsf{SN}(\rightarrow_R)$ implies $=_R$ is decidable.

PROOF: To decide $a =_R b$, just rewrite a and b until you find their normal forms a' and b'. Due to UN (which follows form CR), we have $a =_R b$ iff $a' = b'$.

Corollaries of the Church-Rosser property

THEOREM $\mathsf{CR}(\rightarrow_R)$ implies $\mathsf{UN}(\rightarrow_R)$ (Uniqueness of Normal forms)



If P_1 and P_2 are in normal form, then $P_1 = P_2$, due to CR.

THEOREM $\mathsf{CR}(\rightarrow_R) + \mathsf{SN}(\rightarrow_R)$ implies $=_R$ is decidable.

PROOF: To decide $a =_R b$, just rewrite a and b until you find their normal forms a' and b'. Due to UN (which follows form CR), we have $a =_R b$ iff $a' = b'$.

Foreshadowing: decidability of $=_\beta$ is crucial for decidability of type checking! We will see the conversion rule (next lecture):

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

Parallel reduction in untyped λ-calculus

We prove CR(β) using parallel reduction, a method due to Tait and Martin-Löf and refined by Takahashi.

Parallel reduction $M \Longrightarrow P$ allows to contract several redexes in M in one step. It can be defined inductively.

Parallel reduction in untyped λ-calculus

We prove CR(β) using parallel reduction, a method due to Tait and Martin-Löf and refined by Takahashi.

Parallel reduction $M \Longrightarrow P$ allows to contract several redexes in M in one step. It can be defined inductively.

Definition

$$\frac{M \Longrightarrow M' \qquad P \Longrightarrow P'}{(\lambda x.M)P \Longrightarrow M'[x := P']}\,(\beta) \qquad \frac{M \Longrightarrow M' \qquad P \Longrightarrow P'}{M\,P \Longrightarrow M'\,P'}\,(\text{app})$$

$$\frac{M \Longrightarrow M'}{\lambda x.M \Longrightarrow \lambda x.M'}\,(\lambda) \qquad \frac{}{x \Longrightarrow x}\,(\text{var})$$

Parallel reduction in untyped λ-calculus

We prove CR(β) using parallel reduction, a method due to Tait and Martin-Löf and refined by Takahashi.

Parallel reduction $M \Longrightarrow P$ allows to contract several redexes in M in one step. It can be defined inductively.

DEFINITION

$$\frac{M \Longrightarrow M' \qquad P \Longrightarrow P'}{(\lambda x.M)P \Longrightarrow M'[x := P']}\,(\beta) \qquad \frac{M \Longrightarrow M' \qquad P \Longrightarrow P'}{M\,P \Longrightarrow M'\,P'}\,(\mathrm{app})$$

$$\frac{M \Longrightarrow M'}{\lambda x.M \Longrightarrow \lambda x.M'}\,(\lambda) \qquad \frac{}{x \Longrightarrow x}\,(\mathrm{var})$$

Examples:

$$(\lambda x.y\,x\,x)(\mathbf{I}\,\mathbf{I}) \qquad (\lambda x.x\,(x\,\mathbf{I}))(\mathbf{I}\,\mathbf{I})$$

Properties of parallel reduction

$$\frac{M \Longrightarrow M' \qquad P \Longrightarrow P'}{(\lambda x.M)P \Longrightarrow M'[x := P']}\,(\beta)$$

$$\frac{M \Longrightarrow M' \qquad P \Longrightarrow P'}{M\,P \Longrightarrow M'\,P'}\,(\mathsf{app})$$

$$\frac{M \Longrightarrow M'}{\lambda x.M \Longrightarrow \lambda x.M'}\,(\lambda)$$

$$\frac{}{x \Longrightarrow x}\,(\mathsf{var})$$

Properties of parallel reduction

$$\frac{M \Longrightarrow M' \qquad P \Longrightarrow P'}{(\lambda x.M)P \Longrightarrow M'[x := P']}\,(\beta) \qquad \frac{M \Longrightarrow M' \qquad P \Longrightarrow P'}{M\,P \Longrightarrow M'\,P'}\,(\text{app})$$

$$\frac{M \Longrightarrow M'}{\lambda x.M \Longrightarrow \lambda x.M'}\,(\lambda) \qquad \frac{}{x \Longrightarrow x}\,(\text{var})$$

THEOREM

1. $M \Longrightarrow M$
   The proof is by induction on M.

Properties of parallel reduction

$$\frac{M \Longrightarrow M' \qquad P \Longrightarrow P'}{(\lambda x.M)P \Longrightarrow M'[x := P']}\,(\beta) \qquad \frac{M \Longrightarrow M' \qquad P \Longrightarrow P'}{M\,P \Longrightarrow M'\,P'}\,(\text{app})$$

$$\frac{M \Longrightarrow M'}{\lambda x.M \Longrightarrow \lambda x.M'}\,(\lambda) \qquad \frac{}{x \Longrightarrow x}\,(\text{var})$$

THEOREM

1. $M \Longrightarrow M$
   The proof is by induction on M.
2. If $M \to_\beta P$, then $M \Longrightarrow P$
   The proof is by induction on the derivation, using (1).

Properties of parallel reduction

$$\frac{M \Longrightarrow M' \qquad P \Longrightarrow P'}{(\lambda x.M)P \Longrightarrow M'[x := P']}\,(\beta) \qquad \frac{M \Longrightarrow M' \qquad P \Longrightarrow P'}{M\,P \Longrightarrow M'\,P'}\,(\text{app})$$

$$\frac{M \Longrightarrow M'}{\lambda x.M \Longrightarrow \lambda x.M'}\,(\lambda) \qquad \frac{}{x \Longrightarrow x}\,(\text{var})$$

THEOREM

1. $M \Longrightarrow M$
   The proof is by induction on M.
2. If $M \to_\beta P$, then $M \Longrightarrow P$
   The proof is by induction on the derivation, using (1).
3. If $M \Longrightarrow P$, then $M \twoheadrightarrow_\beta P$.
   The proof is by induction on the derivation.

Parallel reduction satisfies a strong Diamond Property (I)

Theorem

$$\forall M \, \exists Q \, \forall P \, (\text{if } M \Longrightarrow P \text{ then } P \Longrightarrow Q).$$

This immediately implies DP($\Longrightarrow$) (and thereby CR(β)).

Parallel reduction satisfies a strong Diamond Property (I)

THEOREM

$$\forall M\, \exists Q\, \forall P\, (\text{if } M \Longrightarrow P \text{ then } P \Longrightarrow Q).$$

This immediately implies DP($\Longrightarrow$) (and thereby CR(β)).
We can even define this Q inductively from M; it will be called M^*.
So we have

$$\forall M, P\, (\text{if } M \Longrightarrow P \text{ then } P \Longrightarrow M^*).$$

Parallel reduction satisfies a strong Diamond Property (I)

Theorem

$$\forall M \, \exists Q \, \forall P \, (\text{if } M \Longrightarrow P \text{ then } P \Longrightarrow Q).$$

This immediately implies DP($\Longrightarrow$) (and thereby CR(β)).
We can even define this Q inductively from M; it will be called M^*.
So we have

$$\forall M, P \, (\text{if } M \Longrightarrow P \text{ then } P \Longrightarrow M^*).$$

Note: This implies $\forall M \, (M \Longrightarrow M^*)$.

Parallel reduction satisfies a strong Diamond Property (I)

THEOREM

$$\forall M \, \exists Q \, \forall P \, (\text{if } M \Longrightarrow P \text{ then } P \Longrightarrow Q).$$

This immediately implies DP($\Longrightarrow$) (and thereby CR(β)).
We can even define this Q inductively from M; it will be called M^*.
So we have

$$\forall M, P \, (\text{if } M \Longrightarrow P \text{ then } P \Longrightarrow M^*).$$

Note: This implies $\forall M \, (M \Longrightarrow M^*)$.

DEFINITION

$$\begin{array}{rcl} x^* & := & x \\ (\lambda x.M)^* & := & \lambda x.M^* \\ ((\lambda x.P)\, N)^* & := & P^*[x := N^*] \\ (M\, N)^* & := & M^*\, N^* \text{ if } M \neq \lambda x.P \ (M \text{ is not a } \lambda\text{-abstraction}) \end{array}$$

Parallel reduction satisfies a strong Diamond Property (II)

THEOREM

$$\forall M, P\,(\text{if } M \Longrightarrow P \text{ then } P \Longrightarrow M^*).$$

Parallel reduction satisfies a strong Diamond Property (II)

THEOREM

$$\forall M, P\,(\text{if } M \Longrightarrow P \text{ then } P \Longrightarrow M^*).$$

PROOF by induction on the derivation of $M \Longrightarrow P$. There are 4 cases. We treat 3 of them.

Parallel reduction satisfies a strong Diamond Property (II)

THEOREM

$$\forall M, P\,(\text{if } M \Longrightarrow P \text{ then } P \Longrightarrow M^*).$$

PROOF by induction on the derivation of $M \Longrightarrow P$. There are 4 cases. We treat 3 of them.

case (1)

$$\frac{}{x \Longrightarrow x}\,(\text{var})$$

Then indeed $x \Longrightarrow x^*$ (because $x^* = x$).

Parallel reduction satisfies a strong Diamond Property (II)

THEOREM

$$\forall M, P \,(\text{if } M \Longrightarrow P \text{ then } P \Longrightarrow M^*).$$

PROOF by induction on the derivation of $M \Longrightarrow P$. There are 4 cases. We treat 3 of them.

case (1)

$$\frac{}{x \Longrightarrow x}\,(\text{var})$$

Then indeed $x \Longrightarrow x^*$ (because $x^* = x$).

case (2)

$$\frac{M \Longrightarrow M'}{\lambda x.M \Longrightarrow \lambda x.M'}\,(\lambda)$$

Parallel reduction satisfies a strong Diamond Property (II)

THEOREM

$$\forall M, P\,(\text{if } M \Longrightarrow P \text{ then } P \Longrightarrow M^*).$$

PROOF by induction on the derivation of $M \Longrightarrow P$. There are 4 cases. We treat 3 of them.

case (1)

$$\frac{}{x \Longrightarrow x}\,(\text{var})$$

Then indeed $x \Longrightarrow x^*$ (because $x^* = x$).

case (2)

$$\frac{M \Longrightarrow M'}{\lambda x.M \Longrightarrow \lambda x.M'}\,(\lambda)$$

IH: $M' \Longrightarrow M^*$. We need to prove: $\lambda x.M' \Longrightarrow (\lambda x.M)^*$

Parallel reduction satisfies a strong Diamond Property (II)

THEOREM

$$\forall M, P\,(\text{if } M \Longrightarrow P \text{ then } P \Longrightarrow M^*).$$

PROOF by induction on the derivation of $M \Longrightarrow P$. There are 4 cases. We treat 3 of them.

case (1)

$$\frac{}{x \Longrightarrow x}\,(\text{var})$$

Then indeed $x \Longrightarrow x^*$ (because $x^* = x$).

case (2)

$$\frac{M \Longrightarrow M'}{\lambda x.M \Longrightarrow \lambda x.M'}\,(\lambda)$$

IH: $M' \Longrightarrow M^*$. We need to prove: $\lambda x.M' \Longrightarrow (\lambda x.M)^*$
We have $(\lambda x.M)^* = \lambda x.M^*$.
$\lambda x.M' \Longrightarrow \lambda x.M^*$ follows immediately from IH and the definition of $\Longrightarrow$.

Parallel reduction satisfies a strong Diamond Property (III)

THEOREM

$$\forall M, P\,(\text{if } M \Longrightarrow P \text{ then } P \Longrightarrow M^*).$$

PROOF continued

Parallel reduction satisfies a strong Diamond Property (III)

THEOREM

$$\forall M, P \, (\text{if } M \Longrightarrow P \text{ then } P \Longrightarrow M^*).$$

PROOF continued

case (4)

$$\frac{M \Longrightarrow M' \qquad P \Longrightarrow P'}{(\lambda x.M)\, P \Longrightarrow M'[x := P']}$$

Parallel reduction satisfies a strong Diamond Property (III)

THEOREM

$$\forall M, P\,(\text{if } M \Longrightarrow P \text{ then } P \Longrightarrow M^*).$$

PROOF continued

case (4)

$$\frac{M \Longrightarrow M' \qquad P \Longrightarrow P'}{(\lambda x.M)\,P \Longrightarrow M'[x := P']}$$

IH: $M' \Longrightarrow M^*$ and $P' \Longrightarrow P^*$.

Parallel reduction satisfies a strong Diamond Property (III)

THEOREM

$$\forall M, P\,(\text{if } M \Longrightarrow P \text{ then } P \Longrightarrow M^*).$$

PROOF continued

case (4)

$$\frac{M \Longrightarrow M' \qquad P \Longrightarrow P'}{(\lambda x.M)\,P \Longrightarrow M'[x := P']}$$

IH: $M' \Longrightarrow M^*$ and $P' \Longrightarrow P^*$.
We need to prove: $M'[x := P'] \Longrightarrow ((\lambda x.M)\,P)^* = M^*[x := P^*]$.

Parallel reduction satisfies a strong Diamond Property (III)

THEOREM

$$\forall M, P\,(\text{if } M \Longrightarrow P \text{ then } P \Longrightarrow M^*).$$

PROOF continued

case (4)

$$\frac{M \Longrightarrow M' \qquad P \Longrightarrow P'}{(\lambda x.M)\,P \Longrightarrow M'[x := P']}$$

IH: $M' \Longrightarrow M^*$ and $P' \Longrightarrow P^*$.
We need to prove: $M'[x := P'] \Longrightarrow ((\lambda x.M)\,P)^* = M^*[x := P^*]$.

To prove this we need a separate

SUBSTITUTION LEMMA If $M \Longrightarrow M'$ and $P \Longrightarrow P'$, then $M[x := P] \Longrightarrow M'[x := P']$.

This is proved by induction on the structure of M.

DP($\Longrightarrow$) implies CR(β)

The proof that DP($\Longrightarrow$) implies CR(β) follows from the properties we have established:

1. If $M \to_\beta P$, then $M \Longrightarrow P$.
2. If $M \Longrightarrow P$, then $M \twoheadrightarrow_\beta P$.
3. If $M \Longrightarrow P$, then $P \Longrightarrow M^*$.

Example

$$\begin{aligned} x^* &:= x \\ (\lambda x.M)^* &:= \lambda x.M^* \\ (M\,N)^* &:= P^*[x := N^*] \text{ if } M = \lambda x.P \\ &:= M^*\,N^* \text{ otherwise.} \end{aligned}$$

$$(\lambda x.y\,x\,x)(\mathbf{I}\,\mathbf{I})$$

$$(\lambda z.z\,z)\,(\mathbf{I}\,(\mathbf{I}\,x))$$

This is a flexible proof of Church-Rosser

- Methods works for proving CR for reduction in Combinatory Logic
- Methods works for proving CR for β on pseudo-terms of Pure Type Systems

This is a flexible proof of Church-Rosser

- Methods works for proving CR for reduction in Combinatory Logic
- Methods works for proving CR for β on pseudo-terms of Pure Type Systems
- Method extends to typed lambda calculus with data types, for example natural numbers:

$$M, N := x \mid M\,N \mid \lambda x.M \mid 0 \mid \textbf{suc}\,M \mid \textbf{nrec}\,M\,N\,P$$

with

$$\begin{array}{rcl} \textbf{nrec}\,M\,N\,0 & \to & M \\ \textbf{nrec}\,M\,N\,(\textbf{suc}\,P) & \to & N\,P\,(\textbf{nrec}\,M\,N\,P) \end{array}$$

This is a flexible proof of Church-Rosser

- Methods works for proving CR for reduction in Combinatory Logic
- Methods works for proving CR for β on pseudo-terms of Pure Type Systems
- Method extends to typed lambda calculus with data types, for example natural numbers:

$$M, N := x \mid M\,N \mid \lambda x.M \mid 0 \mid \mathbf{suc}\,M \mid \mathbf{nrec}\,M\,N\,P$$

with

$$\begin{array}{rcl} \mathbf{nrec}\,M\,N\,0 & \to & M \\ \mathbf{nrec}\,M\,N\,(\mathbf{suc}\,P) & \to & N\,P\,(\mathbf{nrec}\,M\,N\,P) \end{array}$$

- Method extends to η-reduction:

$$\lambda x.M\,x \to_\eta M \qquad \text{if } x \notin \mathrm{FV}(M)$$

Part 2: Principal Typing

Why do programmers want types?

- Types give a (partial) specification
- Typed terms can't go wrong (Milner)
  Subject Reduction property: If $M : A$ and $M \twoheadrightarrow_\beta N$, then $N : A$.
- Typed terms always terminate
- The type checking algorithm detects (simple) mistakes

But:

Part 2: Principal Typing

Why do programmers want types?

- Types give a (partial) specification
- Typed terms can't go wrong (Milner)
  Subject Reduction property: If $M : A$ and $M \twoheadrightarrow_\beta N$, then $N : A$.
- Typed terms always terminate
- The type checking algorithm detects (simple) mistakes

But:

- The compiler should compute the type information for us! (Why would the programmer have to type all that?)
- This is called a type assignment system, or also typing à la Curry:
- For M an untyped term, the type system assigns a type σ to M (or not)

Simple Type Theory à la Church and à la Curry

$\lambda\!\to$ (à la Church):

$$\frac{x{:}\sigma \in \Gamma}{\Gamma \vdash x : \sigma} \qquad \frac{\Gamma \vdash M : \sigma\to\tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau} \qquad \frac{\Gamma, x{:}\sigma \vdash P : \tau}{\Gamma \vdash \lambda x{:}\sigma.P : \sigma\to\tau}$$

$\lambda\!\to$ (à la Curry):

$$\frac{x{:}\sigma \in \Gamma}{\Gamma \vdash x : \sigma} \qquad \frac{\Gamma \vdash M : \sigma\to\tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau} \qquad \frac{\Gamma, x{:}\sigma \vdash P : \tau}{\Gamma \vdash \lambda x.P : \sigma\to\tau}$$

Type Assignment systems

- With typed assignment also called typing à la Curry, we assign types to untyped λ-terms
$$\lambda x.x : \alpha \to \alpha$$
- As a consequence:
  - Terms do not have unique types,
  - A principal type can be computed using unification.

Type Assignment systems

- With typed assignment also called typing à la Curry, we assign types to untyped λ-terms

$$\lambda x.x : \alpha \rightarrow \alpha$$

- As a consequence:
  - Terms do not have unique types,
  - A principal type can be computed using unification.
- Example:

$$\lambda x.\lambda y.y(\lambda z.x)$$

  can be assigned the types
  - $\alpha \rightarrow ((\beta \rightarrow \alpha) \rightarrow \alpha) \rightarrow \alpha$
  - $(\alpha \rightarrow \alpha) \rightarrow ((\beta \rightarrow \alpha \rightarrow \alpha) \rightarrow \gamma) \rightarrow \gamma$
  - ...

  with $\alpha \rightarrow ((\beta \rightarrow \alpha) \rightarrow \gamma) \rightarrow \gamma$ being the principal type

Example of computing a principal type

Consider $\lambda x.\lambda y.y\,(\lambda z.y\,x)$

Example of computing a principal type

Consider $\lambda x.\lambda y.y\,(\lambda z.y\,x)$

1. Assign type vars to all variables: $x : \alpha, y : \beta, z : \gamma$:

$$\lambda x^{\alpha}.\lambda y^{\beta}.y^{\beta}(\lambda z^{\gamma}.y^{\beta}x^{\alpha})$$

Example of computing a principal type

Consider $\lambda x.\lambda y.y\,(\lambda z.y\,x)$

1. Assign type vars to all variables: $x : \alpha, y : \beta, z : \gamma$:

$$\lambda x^{\alpha}.\lambda y^{\beta}.y^{\beta}(\lambda z^{\gamma}.y^{\beta}x^{\alpha})$$

2. Assign type vars to all applicative subterms: $y\,x$ and $y(\lambda z.y\,x)$:

$$\lambda x^{\alpha}.\lambda y^{\beta}.\underbrace{y^{\beta}(\lambda z^{\gamma}.\overbrace{y^{\beta}x^{\alpha}}^{\delta})}_{\varepsilon}$$

Example of computing a principal type

Consider $\lambda x.\lambda y.y\,(\lambda z.y\,x)$

1. Assign type vars to all variables: $x : \alpha, y : \beta, z : \gamma$:

$$\lambda x^{\alpha}.\lambda y^{\beta}.y^{\beta}(\lambda z^{\gamma}.y^{\beta}x^{\alpha})$$

2. Assign type vars to all applicative subterms: $y\,x$ and $y(\lambda z.y\,x)$:

$$\lambda x^{\alpha}.\lambda y^{\beta}.\underbrace{y^{\beta}(\lambda z^{\gamma}.\overbrace{y^{\beta}x^{\alpha}}^{\delta})}_{\varepsilon}$$

3. Generate equations between types, necessary for the term to be typable:

Example of computing a principal type

Consider $\lambda x.\lambda y.y\,(\lambda z.y\,x)$

1. Assign type vars to all variables: $x : \alpha, y : \beta, z : \gamma$:

$$\lambda x^{\alpha}.\lambda y^{\beta}.y^{\beta}(\lambda z^{\gamma}.y^{\beta}x^{\alpha})$$

2. Assign type vars to all applicative subterms: $y\,x$ and $y(\lambda z.y\,x)$:

$$\lambda x^{\alpha}.\lambda y^{\beta}.\underbrace{y^{\beta}(\lambda z^{\gamma}.\overbrace{y^{\beta}x^{\alpha}}^{\delta})}_{\varepsilon}$$

3. Generate equations between types, necessary for the term to be typable: $\beta = \alpha \to \delta$ $\quad\quad$ $\beta = (\gamma \to \delta) \to \varepsilon$

Example of computing a principal type

Consider $\lambda x.\lambda y.y\,(\lambda z.y\,x)$

1. Assign type vars to all variables: $x : \alpha, y : \beta, z : \gamma$:

$$\lambda x^{\alpha}.\lambda y^{\beta}.y^{\beta}(\lambda z^{\gamma}.y^{\beta}x^{\alpha})$$

2. Assign type vars to all applicative subterms: $y\,x$ and $y(\lambda z.y\,x)$:

$$\lambda x^{\alpha}.\lambda y^{\beta}.\underbrace{y^{\beta}(\lambda z^{\gamma}.\overbrace{y^{\beta}x^{\alpha}}^{\delta})}_{\varepsilon}$$

3. Generate equations between types, necessary for the term to be typable: $\beta = \alpha \to \delta$ $\qquad$ $\beta = (\gamma \to \delta) \to \varepsilon$
4. Find a most general unifier (a substitution) for the type vars that solves the equations: $\alpha := \gamma \to \varepsilon,\ \beta := (\gamma \to \varepsilon) \to \varepsilon,\ \delta := \varepsilon$

Example of computing a principal type

Consider $\lambda x.\lambda y.y\,(\lambda z.y\,x)$

1. Assign type vars to all variables: $x : \alpha, y : \beta, z : \gamma$:

$$\lambda x^\alpha.\lambda y^\beta.y^\beta(\lambda z^\gamma.y^\beta x^\alpha)$$

2. Assign type vars to all applicative subterms: $y\,x$ and $y(\lambda z.y\,x)$:

$$\lambda x^\alpha.\lambda y^\beta.\underbrace{y^\beta(\lambda z^\gamma.\overbrace{y^\beta x^\alpha}^{\delta})}_{\varepsilon}$$

3. Generate equations between types, necessary for the term to be typable: $\beta = \alpha \to \delta$ $\qquad$ $\beta = (\gamma \to \delta) \to \varepsilon$
4. Find a most general unifier (a substitution) for the type vars that solves the equations: $\alpha := \gamma \to \varepsilon,\ \beta := (\gamma \to \varepsilon) \to \varepsilon,\ \delta := \varepsilon$
5. The principal type of $\lambda x.\lambda y.y(\lambda z.yx)$ is now

$$(\gamma \to \varepsilon) \to ((\gamma \to \varepsilon) \to \varepsilon) \to \varepsilon$$

Example of computing a principal type (II)

$$\lambda x.\lambda y.x\,(y\,x)$$

Which of these terms is typable?

- $M_1 := \lambda x.x\,(\lambda y.y\,x)$
- $M_2 := \lambda x.\lambda y.x\,(x\,y)$
- $M_3 := \lambda x.\lambda y.x\,(\lambda z.y\,x)$

Which of these terms is typable?

- $M_1 := \lambda x.x\,(\lambda y.y\,x)$
- $M_2 := \lambda x.\lambda y.x\,(x\,y)$
- $M_3 := \lambda x.\lambda y.x\,(\lambda z.y\,x)$

Poll:

A M_1 is not typable, M_2 and M_3 are typable.
B M_2 is not typable, M_1 and M_3 are typable.
C M_3 is not typable, M_1 and M_2 are typable.

Principal Types: Definitions

- A type substitution (or just substitution) is a map S from type variables to types with a finite domain such that variables that occur in the range of S are not in the domain of S.
- A substitution S is written as $[\alpha_1 := \sigma_1, \ldots, \alpha_n := \sigma_n]$ where
  - all α_i are different
  - $\alpha_i \notin \sigma_j$ (for all i, j).

Principal Types: Definitions

- A type substitution (or just substitution) is a map S from type variables to types with a finite domain such that variables that occur in the range of S are not in the domain of S.
- A substitution S is written as $[\alpha_1 := \sigma_1, \ldots, \alpha_n := \sigma_n]$ where
  - all α_i are different
  - $\alpha_i \notin \sigma_j$ (for all i, j).
- We write $\tau\, S$ for substitution S applied to τ.
- We can compose substitutions: $S; T$. (So we have $\tau\,(S; T) = (\tau\, S)\, T$.)

Principal Types: Definitions

- A type substitution (or just substitution) is a map S from type variables to types with a finite domain such that variables that occur in the range of S are not in the domain of S.
- A substitution S is written as $[\alpha_1 := \sigma_1, \ldots, \alpha_n := \sigma_n]$ where
  - all α_i are different
  - $\alpha_i \notin \sigma_j$ (for all i, j).
- We write $\tau\, S$ for substitution S applied to τ.
- We can compose substitutions: $S; T$. (So we have $\tau\,(S; T) = (\tau\, S)\, T$.)
- A unifier of the types σ and τ is a substitution that solves $\sigma = \tau$, i.e. an S such that $\sigma\, S = \tau\, S$

Principal Types: Definitions

- A type substitution (or just substitution) is a map S from type variables to types with a finite domain such that variables that occur in the range of S are not in the domain of S.
- A substitution S is written as $[\alpha_1 := \sigma_1, \ldots, \alpha_n := \sigma_n]$ where
  - all α_i are different
  - $\alpha_i \notin \sigma_j$ (for all i, j).
- We write $\tau\, S$ for substitution S applied to τ.
- We can compose substitutions: $S; T$. (So we have $\tau\,(S; T) = (\tau\, S)\, T$.)
- A unifier of the types σ and τ is a substitution that solves $\sigma = \tau$, i.e. an S such that $\sigma\, S = \tau\, S$
- A most general unifier (mgu) of the types σ and τ is the "simplest substitution" that solves $\sigma = \tau$, i.e. an S such that
  - $\sigma\, S = \tau\, S$
  - for all substitutions T such that $\sigma\, T = \tau\, T$ there is a substitution R such that $T = S; R$.

Principal Types: solving a list of equations

All notions generalize to lists of equations

$$\mathcal{E} = \langle \sigma_1 = \tau_1, \ldots, \sigma_n = \tau_n \rangle$$

instead of a single equation $\sigma = \tau$.

- A unifier of $\mathcal{E}$ is a substitution S such that $\sigma_i\, S = \tau_i\, S$ for all i.

Principal Types: solving a list of equations

All notions generalize to lists of equations

$$\mathcal{E} = \langle \sigma_1 = \tau_1, \ldots, \sigma_n = \tau_n \rangle$$

instead of a single equation $\sigma = \tau$.

- A unifier of $\mathcal{E}$ is a substitution S such that $\sigma_i\, S = \tau_i\, S$ for all i.
- A most general unifier (mgu) for $\mathcal{E}$ is an S such that
  - $\sigma_i\, S = \tau_i\, S$ for all i
  - for all substitutions T such that $\sigma_i\, T = \tau_i\, T$ for all i, there is a substitution R such that $T = S; R$.

Computability of most general unifiers

THEOREM There is an algorithm U that, given a list of equations $\mathcal{E} = \langle \sigma_1 = \tau_1, \ldots, \sigma_n = \tau_n \rangle$ outputs

- A most general unifier of $\mathcal{E}$ if these equations can be solved.
- "Fail" if $\mathcal{E}$ can't be solved.

Computability of most general unifiers

THEOREM There is an algorithm U that, given a list of equations $\mathcal{E} = \langle \sigma_1 = \tau_1, \ldots, \sigma_n = \tau_n \rangle$ outputs

- A most general unifier of $\mathcal{E}$ if these equations can be solved.
- "Fail" if $\mathcal{E}$ can't be solved.

PROOF

- $U(\langle \alpha = \alpha, \ldots, \sigma_n = \tau_n \rangle) := U(\langle \sigma_2 = \tau_2, \ldots, \sigma_n = \tau_n \rangle)$.
- $U(\langle \alpha = \tau_1, \ldots, \sigma_n = \tau_n \rangle) :=$ "Fail" if $\alpha \in \mathsf{FV}(\tau_1)$, $\tau_1 \neq \alpha$.
- $U(\langle \sigma_1 = \alpha, \ldots, \sigma_n = \tau_n \rangle) := U(\langle \alpha = \sigma_1, \ldots, \sigma_n = \tau_n \rangle)$

Computability of most general unifiers

THEOREM There is an algorithm U that, given a list of equations $\mathcal{E} = \langle \sigma_1 = \tau_1, \ldots, \sigma_n = \tau_n \rangle$ outputs

- A most general unifier of $\mathcal{E}$ if these equations can be solved.
- "Fail" if $\mathcal{E}$ can't be solved.

PROOF

- $U(\langle \alpha = \alpha, \ldots, \sigma_n = \tau_n \rangle) := U(\langle \sigma_2 = \tau_2, \ldots, \sigma_n = \tau_n \rangle)$.
- $U(\langle \alpha = \tau_1, \ldots, \sigma_n = \tau_n \rangle) :=$ "Fail" if $\alpha \in \mathrm{FV}(\tau_1)$, $\tau_1 \neq \alpha$.
- $U(\langle \sigma_1 = \alpha, \ldots, \sigma_n = \tau_n \rangle) := U(\langle \alpha = \sigma_1, \ldots, \sigma_n = \tau_n \rangle)$
- $U(\langle \alpha = \tau_1, \ldots, \sigma_n = \tau_n \rangle) := [\alpha := V(\tau_1), V]$, if $\alpha \notin \mathrm{FV}(\tau_1)$, where V abbreviates $U(\langle \sigma_2[\alpha := \tau_1] = \tau_2[\alpha := \tau_1], \ldots, \sigma_n[\alpha := \tau_1] = \tau_n[\alpha := \tau_1] \rangle)$.

Computability of most general unifiers

THEOREM There is an algorithm U that, given a list of equations $\mathcal{E} = \langle \sigma_1 = \tau_1, \ldots, \sigma_n = \tau_n \rangle$ outputs

- A most general unifier of $\mathcal{E}$ if these equations can be solved.
- "Fail" if $\mathcal{E}$ can't be solved.

PROOF

- $U(\langle \alpha = \alpha, \ldots, \sigma_n = \tau_n \rangle) := U(\langle \sigma_2 = \tau_2, \ldots, \sigma_n = \tau_n \rangle)$.
- $U(\langle \alpha = \tau_1, \ldots, \sigma_n = \tau_n \rangle) :=$ "Fail" if $\alpha \in \mathrm{FV}(\tau_1)$, $\tau_1 \neq \alpha$.
- $U(\langle \sigma_1 = \alpha, \ldots, \sigma_n = \tau_n \rangle) := U(\langle \alpha = \sigma_1, \ldots, \sigma_n = \tau_n \rangle)$
- $U(\langle \alpha = \tau_1, \ldots, \sigma_n = \tau_n \rangle) := [\alpha := V(\tau_1), V]$, if $\alpha \notin \mathrm{FV}(\tau_1)$, where V abbreviates $U(\langle \sigma_2[\alpha := \tau_1] = \tau_2[\alpha := \tau_1], \ldots, \sigma_n[\alpha := \tau_1] = \tau_n[\alpha := \tau_1] \rangle)$.
- $U(\langle \mu \to \nu = \rho \to \xi, \ldots, \sigma_n = \tau_n \rangle) :=$
  $U(\langle \mu = \rho, \nu = \xi, \ldots, \sigma_n = \tau_n \rangle)$

Principal type

DEFINITION σ is a principal type for the untyped closed λ-term M if

- $\vdash M : \sigma$ in $\lambda\!\rightarrow$ à la Curry
- for all types τ, if $\vdash M : \tau$, then $\tau = \sigma\, T$ for some substitution T.

Principal Type Theorem

THEOREM There is an algorithm PT that, when given an (untyped) closed λ-term M, outputs

- A principal type σ for M if M is typable in $\lambda\!\rightarrow$ à la Curry.
- "Fail" if M is not typable in $\lambda\!\rightarrow$ à la Curry.

Principal Type Theorem

THEOREM There is an algorithm PT that, when given an (untyped) closed λ-term M, outputs

- A principal type σ for M if M is typable in $\lambda\!\to$ à la Curry.
- "Fail" if M is not typable in $\lambda\!\to$ à la Curry.

PROOF In the algorithm we

- first label the bound variables and all applicative sub-terms with type variables, and we give the candidate type τ,
- then we generate the equations that need to hold for the term to be typable,
- then we compute the mgu of this set of equations and we obtain the substitution S or "Fail",
- then we have as output the principal type $\tau\,S$ or "Fail".

The proof that this output indeed correctly computes the principal type can be found in the literature.

Conclusion

Today we saw

- A proof of the Church-Rosser property for the untyped λ-calculus
- Key technique: **parallel reduction**
- We also an algorithm to assign types to untyped λ-terms
- Important: this algorithm finds **principal types**