- Lecture 1: Introduction, Abstract Rewriting
- Lecture 2: Term Rewriting
- Lecture 3: Combinatory Logic
- Lecture 4: Termination
- Lecture 5: Matching, Unification
- Lecture 6: Equational Reasoning, Completion
- Lecture 7: Confluence
- Lecture 8: Modularity
- Lecture 9: Strategies
- Lecture 10: Decidability
- Lecture 11: Infinitary Rewriting

Outline

- Overview
- Combinatory Logic

Combinatory Logic

Combinatory Logic (CL)

$$
\begin{array}{lcl}
Ap(Ap(Ap(S,x),y),z) & \rightarrow & Ap(Ap(x,z),Ap(y,z)) \\
Ap(Ap(K,x),y) & \rightarrow & x \\
Ap(I,x) & \rightarrow & x
\end{array}
$$

Combinatory Logic (CL)

$$
\begin{array}{lcl}
Ap(Ap(Ap(S,x),y),z) & \rightarrow & Ap(Ap(x,z),Ap(y,z)) \\
Ap(Ap(K,x),y) & \rightarrow & x \\
Ap(I,x) & \rightarrow & x
\end{array}
$$

CL in infix notation

$$
\begin{array}{lcl}
(((S \cdot x) \cdot y) \cdot z) & \rightarrow & ((x \cdot z) \cdot (y \cdot z)) \\
((K \cdot x) \cdot y) & \rightarrow & x \\
(I \cdot x) & \rightarrow & x
\end{array}
$$

Combinatory Logic (CL)

$$\begin{array}{lcl} Ap(Ap(Ap(S,x),y),z) & \rightarrow & Ap(Ap(x,z),Ap(y,z)) \\ Ap(Ap(K,x),y) & \rightarrow & x \\ Ap(I,x) & \rightarrow & x \end{array}$$

CL in infix notation

$$\begin{array}{lcl} (((S \cdot x) \cdot y) \cdot z) & \rightarrow & ((x \cdot z) \cdot (y \cdot z)) \\ ((K \cdot x) \cdot y) & \rightarrow & x \\ (I \cdot x) & \rightarrow & x \end{array}$$

CL in standard notation

$$\begin{array}{lcl} Sxyz & \rightarrow & xz(yz) \\ Kxy & \rightarrow & x \\ Ix & \rightarrow & x \end{array}$$

Association to the Left

Association to the Left

A term $t_1 t_2 t_3 \ldots t_n$ restores to $((\ldots((t_1 t_2)t_3)\ldots)t_n)$

Association to the Left

Association to the Left

A term $t_1 t_2 t_3 \ldots t_n$ restores to $((\ldots((t_1 t_2)t_3)\ldots)t_n)$

- $xz(yz)$ restores to $(xz)(yz)$
  not to $x(z(yz))$

Association to the Left

Association to the Left

A term $t_1 t_2 t_3 \ldots t_n$ restores to $((\ldots((t_1 t_2)t_3)\ldots)t_n)$

- $xz(yz)$ restores to $(xz)(yz)$
  not to $x(z(yz))$
- Kxy restores to $(Kx)y$
  not $K(xy)$

Association to the Left

Association to the Left

A term $t_1 t_2 t_3 \ldots t_n$ restores to $((\ldots((t_1 t_2)t_3)\ldots)t_n)$

- $xz(yz)$ restores to $(xz)(yz)$
  not to $x(z(yz))$
- Kxy restores to $(Kx)y$
  not $K(xy)$
- Not all bracket pairs can be dropped:
  $xzyz$ is when restored $((xz)y)z$
  quite different from $xz(yz)$

Association to the Left

Association to the Left

A term $t_1 t_2 t_3 \ldots t_n$ restores to $((\ldots((t_1 t_2)t_3)\ldots)t_n)$

- $xz(yz)$ restores to $(xz)(yz)$
  not to $x(z(yz))$
- Kxy restores to $(Kx)y$
  not $K(xy)$
- Not all bracket pairs can be dropped:
  $xzyz$ is when restored $((xz)y)z$
  quite different from $xz(yz)$
- Note that the term SIx does not contain a redex Ix.

A Famous Term

- A famous term with a famous reduction cycle:

$$SII(SII)$$

A Famous Term

- A famous term with a famous reduction cycle:

$$SII(SII) \rightarrow I(SII)(I(SII))$$

A Famous Term

- A famous term with a famous reduction cycle:

$$\begin{aligned} SII(SII) &\to I(SII)(I(SII)) \\ &\to SII(I(SII)) \end{aligned}$$

A Famous Term

- A famous term with a famous reduction cycle:

$$\begin{aligned} SII(SII) &\to I(SII)(I(SII)) \\ &\to SII(I(SII)) \\ &\to SII(SII) \end{aligned}$$

A Famous Term

- A famous term with a famous reduction cycle:

$$\begin{aligned} SII(SII) &\to I(SII)(I(SII)) \\ &\to SII(I(SII)) \\ &\to SII(SII) \end{aligned}$$

- Let $D = SII$.

  Given an arbitrary argument, D copies it and applies it to itself:

$$Dx = SIIx$$

A Famous Term

- A famous term with a famous reduction cycle:

$$\begin{aligned} SII(SII) &\to I(SII)(I(SII)) \\ &\to SII(I(SII)) \\ &\to SII(SII) \end{aligned}$$

- Let $D = SII$.

  Given an arbitrary argument, D copies it and applies it to itself:

$$Dx = SIIx \to Ix(Ix)$$

A Famous Term

- A famous term with a famous reduction cycle:

$$\begin{aligned} SII(SII) &\to I(SII)(I(SII)) \\ &\to SII(I(SII)) \\ &\to SII(SII) \end{aligned}$$

- Let $D = SII$.

  Given an arbitrary argument, D copies it and applies it to itself:

$$\begin{aligned} Dx = SIIx &\to Ix(Ix) \\ &\to x(Ix) \end{aligned}$$

A Famous Term

- A famous term with a famous reduction cycle:

$$\begin{aligned} SII(SII) &\to I(SII)(I(SII)) \\ &\to SII(I(SII)) \\ &\to SII(SII) \end{aligned}$$

- Let $D = SII$.

  Given an arbitrary argument, D copies it and applies it to itself:

$$\begin{aligned} Dx = SIIx &\to Ix(Ix) \\ &\to x(Ix) \\ &\to xx \end{aligned}$$

Combinators

- Let $B = S(KS)K$.

  We have

$$Bxyz = S(KS)Kxyz$$

Combinators

- Let $B = S(KS)K$.

  We have

$$Bxyz = S(KS)Kxyz \rightarrow KSx(Kx)yz$$

Combinators

- Let $B = S(KS)K$.

  We have

$$\begin{aligned} Bxyz = S(KS)Kxyz &\to KSx(Kx)yz \\ &\to S(Kx)yz \end{aligned}$$

Combinators

- Let $B = S(KS)K$.

  We have

$$\begin{aligned} Bxyz = S(KS)Kxyz &\to KSx(Kx)yz \\ &\to S(Kx)yz \\ &\to Kxz(yz) \end{aligned}$$

Combinators

- Let $B = S(KS)K$.

  We have

$$\begin{aligned} Bxyz = S(KS)Kxyz &\to KSx(Kx)yz \\ &\to S(Kx)yz \\ &\to Kxz(yz) \\ &\to x(yz) \end{aligned}$$

Combinators

- Let $B = S(KS)K$.

  We have

$$\begin{aligned} Bxyz = S(KS)Kxyz &\to KSx(Kx)yz \\ &\to S(Kx)yz \\ &\to Kxz(yz) \\ &\to x(yz) \end{aligned}$$

- Let $C = S(BBS)(KK)$.

  We have

$$Cxyz \to^* xzy$$

Combinators

- Let $B = S(KS)K$.

  We have

$$\begin{aligned} Bxyz = S(KS)Kxyz &\to KSx(Kx)yz \\ &\to S(Kx)yz \\ &\to Kxz(yz) \\ &\to x(yz) \end{aligned}$$

- Let $C = S(BBS)(KK)$.

  We have

$$Cxyz \to^* xzy$$

- Exercise: find a combinator F such that $Fxy = yx$.

Combinatorial Completeness

Lemma (Combinatorial Completeness)

Given CL-term t, one can find a CL-term F without variables such that

$$Fx_1 \dots x_n \to^* t$$

Combinatorial Completeness

Lemma (Combinatorial Completeness)

Given CL-term t, one can find a CL-term F without variables such that

$$Fx_1 \dots x_n \to^* t$$

Lemma

Then $F\ t_1 \dots t_n \to^* t[x_1 \mapsto t_1, \dots x_n \mapsto t_n]$ for arbitrary terms $t_1, \dots, t_n$.

Combinatorial Completeness

Lemma (Combinatorial Completeness)

Given CL-term t, one can find a CL-term F without variables such that

$$Fx_1 \dots x_n \to^* t$$

Then closure under substitutions yields:

Lemma

Then $F\ t_1 \dots t_n \to^* t[x_1 \mapsto t_1, \dots x_n \mapsto t_n]$ for arbitrary terms $t_1, \dots, t_n$.

Towards a Proof of Combinatorial Completeness

Definition (Abstraction of x)

1. $[x]t = Kt$, if t is a constant or a variable other than x
2. $[x]x = I$
3. $[x]tt' = S([x]t)([x]t')$.

$= S([y]y)([y]x) = SI(Kx)$,

- $t' = [x]t = [x](SI(Kx)) = S([x](SI))([x](Kx))$
  $= S(K(SI))(S([x]K)([x]x)) = S(K(SI))(S(KK)I)$.

Towards a Proof of Combinatorial Completeness

Definition (Abstraction of x)

1. $[x]t = Kt$, if t is a constant or a variable other than x
2. $[x]x = I$
3. $[x]tt' = S([x]t)([x]t')$.

For $[x_1]([x_2](\ldots([x_n]t)\ldots))$ we will write $[x_1x_2\ldots x_n]t$

$= S([y]y)([y]x) = SI(Kx)$,

- $t' = [x]t = [x](SI(Kx)) = S([x](SI))([x](Kx))$
  $= S(K(SI))(S([x]K)([x]x)) = S(K(SI))(S(KK)I)$.

Towards a Proof of Combinatorial Completeness

Definition (Abstraction of x)

1. $[x]t = Kt$, if t is a constant or a variable other than x
2. $[x]x = I$
3. $[x]tt' = S([x]t)([x]t')$.

For $[x_1]([x_2](\ldots([x_n]t)\ldots))$ we will write $[x_1x_2\ldots x_n]t$

Hint: $[x]t = Kt$ can also be used if t does not contain x.

$= S([y]y)([y]x) = SI(Kx)$,

- $t' = [x]t = [x](SI(Kx)) = S([x](SI))([x](Kx))$
  $= S(K(SI))(S([x]K)([x]x)) = S(K(SI))(S(KK)I)$.

Towards a Proof of Combinatorial Completeness

Definition (Abstraction of x)

1. $[x]t = Kt$, if t is a constant or a variable other than x
2. $[x]x = I$
3. $[x]tt' = S([x]t)([x]t')$.

For $[x_1]([x_2](\ldots([x_n]t)\ldots))$ we will write $[x_1x_2\ldots x_n]t$

Hint: $[x]t = Kt$ can also be used if t does not contain x.

Example

Let $t = [y]yx$ and $t' = [xy]yx$. Then

1. t

Towards a Proof of Combinatorial Completeness

Definition (Abstraction of x)

1. $[x]t = Kt$, if t is a constant or a variable other than x
2. $[x]x = I$
3. $[x]tt' = S([x]t)([x]t')$.

For $[x_1]([x_2](\ldots([x_n]t)\ldots))$ we will write $[x_1x_2\ldots x_n]t$

Hint: $[x]t = Kt$ can also be used if t does not contain x.

Example

Let $t = [y]yx$ and $t' = [xy]yx$. Then

1. $t = S([y]y)([y]x)$

Towards a Proof of Combinatorial Completeness

Definition (Abstraction of x)

1. $[x]t = Kt$, if t is a constant or a variable other than x
2. $[x]x = I$
3. $[x]tt' = S([x]t)([x]t')$.

For $[x_1]([x_2](\ldots([x_n]t)\ldots))$ we will write $[x_1x_2\ldots x_n]t$

Hint: $[x]t = Kt$ can also be used if t does not contain x.

Example

Let $t = [y]yx$ and $t' = [xy]yx$. Then

1. $t = S([y]y)([y]x) = SI(Kx)$,

Towards a Proof of Combinatorial Completeness

Definition (Abstraction of x)

1. $[x]t = Kt$, if t is a constant or a variable other than x
2. $[x]x = I$
3. $[x]tt' = S([x]t)([x]t')$.

For $[x_1]([x_2](\ldots([x_n]t)\ldots))$ we will write $[x_1x_2\ldots x_n]t$

Hint: $[x]t = Kt$ can also be used if t does not contain x.

Example

Let $t = [y]yx$ and $t' = [xy]yx$. Then

1. $t = S([y]y)([y]x) = SI(Kx)$,
2. t'

Towards a Proof of Combinatorial Completeness

Definition (Abstraction of x)

1. $[x]t = Kt$, if t is a constant or a variable other than x
2. $[x]x = I$
3. $[x]tt' = S([x]t)([x]t')$.

For $[x_1]([x_2](\ldots([x_n]t)\ldots))$ we will write $[x_1x_2\ldots x_n]t$

Hint: $[x]t = Kt$ can also be used if t does not contain x.

Example

Let $t = [y]yx$ and $t' = [xy]yx$. Then

1. $t = S([y]y)([y]x) = SI(Kx)$,
2. $t' = [x]t$

Towards a Proof of Combinatorial Completeness

Definition (Abstraction of x)

1. $[x]t = Kt$, if t is a constant or a variable other than x
2. $[x]x = I$
3. $[x]tt' = S([x]t)([x]t')$.

For $[x_1]([x_2](\ldots([x_n]t)\ldots))$ we will write $[x_1x_2\ldots x_n]t$

Hint: $[x]t = Kt$ can also be used if t does not contain x.

Example

Let $t = [y]yx$ and $t' = [xy]yx$. Then

1. $t = S([y]y)([y]x) = SI(Kx)$,
2. $t' = [x]t = [x](SI(Kx))$

Towards a Proof of Combinatorial Completeness

Definition (Abstraction of x)

1. $[x]t = Kt$, if t is a constant or a variable other than x
2. $[x]x = I$
3. $[x]tt' = S([x]t)([x]t')$.

For $[x_1]([x_2](\ldots([x_n]t)\ldots))$ we will write $[x_1x_2\ldots x_n]t$

Hint: $[x]t = Kt$ can also be used if t does not contain x.

Example

Let $t = [y]yx$ and $t' = [xy]yx$. Then

1. $t = S([y]y)([y]x) = SI(Kx)$,
2. $t' = [x]t = [x](SI(Kx)) = S([x](SI))([x](Kx))$

Towards a Proof of Combinatorial Completeness

Definition (Abstraction of x)

1. $[x]t = Kt$, if t is a constant or a variable other than x
2. $[x]x = I$
3. $[x]tt' = S([x]t)([x]t')$.

For $[x_1]([x_2](\ldots([x_n]t)\ldots))$ we will write $[x_1x_2\ldots x_n]t$

Hint: $[x]t = Kt$ can also be used if t does not contain x.

Example

Let $t = [y]yx$ and $t' = [xy]yx$. Then

1. $t = S([y]y)([y]x) = SI(Kx)$,
2. $t' = [x]t = [x](SI(Kx)) = S([x](SI))([x](Kx))$
   $= S(K(SI))(S([x]K)([x]x))$

Towards a Proof of Combinatorial Completeness

Definition (Abstraction of x)

1. $[x]t = Kt$, if t is a constant or a variable other than x
2. $[x]x = I$
3. $[x]tt' = S([x]t)([x]t')$.

For $[x_1]([x_2](\ldots([x_n]t)\ldots))$ we will write $[x_1x_2\ldots x_n]t$

Hint: $[x]t = Kt$ can also be used if t does not contain x.

Example

Let $t = [y]yx$ and $t' = [xy]yx$. Then

1. $t = S([y]y)([y]x) = SI(Kx)$,
2. $t' = [x]t = [x](SI(Kx)) = S([x](SI))([x](Kx))$
   $= S(K(SI))(S([x]K)([x]x)) = S(K(SI))(S(KK)I)$.

Towards a Proof of Combinatorial Completeness

Lemma (Properties of)

1. $([x]t)x \to^* t$
2. *The variable x does not occur in the CL-term denoted by $[x]t$*

Proof.

Induction on t:

base case $([x]x)x = Ix \to x$

$([x]y)x = Kyx \to y$ (the same if t is a constant)

Towards a Proof of Combinatorial Completeness

Lemma (Properties of)

1. $([x]t)x \to^* t$
2. *The variable x does not occur in the CL-term denoted by $[x]t$*

Proof.

Induction on t:

base case $([x]x)x = Ix \to x$

$([x]y)x = Kyx \to y$ (the same if t is a constant)

Towards a Proof of Combinatorial Completeness

Lemma (Properties of)

1. $([x]t)x \to^* t$
2. *The variable x does not occur in the CL-term denoted by $[x]t$*

Proof.

Induction on t:

base case $([x]x)x = Ix \to x$

$([x]y)x = Kyx \to y$ (the same if t is a constant)

induction IH: $([x]t)x \to^* t$ and $([x]t')x \to^* t'$

Towards a Proof of Combinatorial Completeness

Lemma (Properties of)

1. $([x]t)x \to^* t$
2. *The variable x does not occur in the CL-term denoted by $[x]t$*

Proof.

Induction on t:

base case $([x]x)x = Ix \to x$

$([x]y)x = Kyx \to y$ (the same if t is a constant)

induction IH: $([x]t)x \to^* t$ and $([x]t')x \to^* t'$

$$\begin{aligned}([x]tt')x &= S([x]t)([x]t')x \to ([x]t)x(([x]t')x) \\ &\to^* t(([x]t')x) \to^* tt'\end{aligned}$$

Simulation of beta reduction

Lemma

We can use abstraction to simulate β-reduction of λ-calculus:

$$([x]t)t' \to^* t[x := t']$$

Simulation of beta reduction

Lemma

We can use abstraction to simulate β-reduction of λ-calculus:

$$([x]t)t' \to^* t[x := t']$$

Proof.

We have $([x]t)x \to^* t$.
Substitute t' for x in this reduction. ■

Proof of Combinatorial Completeness

Combinatorial Completeness

Given CL-term t, one can find a CL-term F without variables such that

$$Fx_1 \dots x_n \to^* t$$

By former proposition and induction on n:

$$Fx_1 \dots x_n = ([x_1][x_2 \dots x_n]t)x_1 \dots x_n \to^* ([x_2 \dots x_n]t)x_2 \dots x_n \to^* t$$

Proof of Combinatorial Completeness

Combinatorial Completeness

Given CL-term t, one can find a CL-term F without variables such that

$$Fx_1 \dots x_n \to^* t$$

Proof.

Let $x_1, \dots, x_n$ contain all variables in t (possibly more). Let

$$F = [x_1 x_2 \dots x_n]t$$

Proof of Combinatorial Completeness

Combinatorial Completeness

Given CL-term t, one can find a CL-term F without variables such that

$$Fx_1 \dots x_n \to^* t$$

Proof.

Let $x_1, \dots, x_n$ contain all variables in t (possibly more). Let

$$F = [x_1 x_2 \dots x_n]t$$

By former proposition and induction on n:

$$Fx_1 \dots x_n = ([x_1][x_2 \dots x_n]t)x_1 \dots x_n \to^* ([x_2 \dots x_n]t)x_2 \dots x_n \to^* t$$

Fixed Points

Let F be an arbitrary CL-term. Consider:

$$P_F = D(BFD)$$

Then $F(PF) = PF$ for any F.

Fixed Points

Let F be an arbitrary CL-term. Consider:

$$P_F = D(BFD)$$

$$P_F = D(BFD)$$

Then $F(PF) = PF$ for any F.

Fixed Points

Let F be an arbitrary CL-term. Consider:

$$P_F = D(BFD)$$

$$P_F = D(BFD) \rightarrow^* BFD(BFD)$$

Then $F(PF) = PF$ for any F.

Fixed Points

Let F be an arbitrary CL-term. Consider:

$$P_F = D(BFD)$$

$$P_F = D(BFD) \rightarrow^* BFD(BFD) \rightarrow^* F(D(BFD)) = FP_F$$

Then $F(PF) = PF$ for any F.

Fixed Points

Let F be an arbitrary CL-term. Consider:

$$P_F = D(BFD)$$

$$P_F = D(BFD) \rightarrow^* BFD(BFD) \rightarrow^* F(D(BFD)) = FP_F$$

Hence $FP_F \leftrightarrow^* P_F$.

Then $F(PF) = PF$ for any F.

Fixed Points

Let F be an arbitrary CL-term. Consider:

$$P_F = D(BFD)$$

$$P_F = D(BFD) \to^* BFD(BFD) \to^* F(D(BFD)) = FP_F$$

Hence $FP_F \leftrightarrow^* P_F$. Looks better if we write $=$ for $\leftrightarrow^*$:

$$FP_F = P_F$$

Then $F(PF) = PF$ for any F.

Fixed Points

Let F be an arbitrary CL-term. Consider:

$$P_F = D(BFD)$$

$$P_F = D(BFD) \to^* BFD(BFD) \to^* F(D(BFD)) = FP_F$$

Hence $FP_F \leftrightarrow^* P_F$. Looks better if we write $=$ for $\leftrightarrow^*$:

$$FP_F = P_F$$

P_F is a *fixed point* for F

Then $F(PF) = PF$ for any F.

Fixed Points

Let F be an arbitrary CL-term. Consider:

$$P_F = D(BFD)$$

$$P_F = D(BFD) \to^* BFD(BFD) \to^* F(D(BFD)) = FP_F$$

Hence $FP_F \leftrightarrow^* P_F$. Looks better if we write $=$ for $\leftrightarrow^*$:

$$FP_F = P_F$$

P_F is a *fixed point* for F

Define the *fixed-point combinator* $P = [x]D(BxD)$.

Fixed Points

Let F be an arbitrary CL-term. Consider:

$$P_F = D(BFD)$$

$$P_F = D(BFD) \to^* BFD(BFD) \to^* F(D(BFD)) = FP_F$$

Hence $FP_F \leftrightarrow^* P_F$. Looks better if we write $=$ for $\leftrightarrow^*$:

$$FP_F = P_F$$

P_F is a *fixed point* for F

Define the *fixed-point combinator* $P = [x]D(BxD)$. Then $F(PF) = PF$ for any F.

Fixed-point Combinators

Definition

A fixed-point combinator Y is any closed CL-term for which there is a conversion

$$Yx \leftrightarrow^* x(Yx)$$

Fixed-point Combinators

Definition

A fixed-point combinator Y is any closed CL-term for which there is a conversion

$$Yx \leftrightarrow^* x(Yx)$$

Many fixed-point combinators exist in CL.

The most famous one is Curry's: *paradoxical combinator*

$$Y_C = SSI(SB(KD))$$

Implicit function definition

Given a CL-term t, find F such that

$$Fx_1 \dots x_n \leftrightarrow^* t[y := F]$$

Implicit function definition

Given a CL-term t, find F such that

$$Fx_1 \dots x_n \leftrightarrow^* t[y := F]$$

Application: recursion.

Define functions like: $F\ x =$ "if $x = 0$ then 1 else $x \cdot F\ (x - 1)$"

Implicit function definition

Given a CL-term t, find F such that

$$Fx_1 \dots x_n \ \leftrightarrow^* \ t[y := F]$$

Application: recursion.

Define functions like: $F\ x =$ "if $x = 0$ then 1 else $x \cdot F\ (x - 1)$"

We take:

- $t' = [y][x_1 \dots x_n]t$, and
- $F = Yt'$ for some fixed-point combinator Y.

Implicit function definition

Given a CL-term t, find F such that

$$Fx_1 \dots x_n \leftrightarrow^* t[y := F]$$

Application: recursion.

Define functions like: $F\ x =$ "if $x = 0$ then 1 else $x \cdot F\ (x - 1)$"

We take:

- $t' = [y][x_1 \dots x_n]t$, and
- $F = Yt'$ for some fixed-point combinator Y.

Then:

$$Fx_1 \dots x_n = t'Fx_1 \dots x_n = t'yx_1 \dots x_n[y := F] \rightarrow^* t[y := F]$$

Church Booleans

Church encoding of boolean values *true* and *false*:

$$\text{true} = K$$

Church Booleans

Church encoding of boolean values *true* and *false*:

$$\text{true} = K \qquad \text{true}\ x\ y \to^* x$$

$$\text{false} = KI$$

Church Booleans

Church encoding of boolean values *true* and *false*:

$$\text{true} = K \qquad \text{true}\ x\ y \to^* x$$

$$\text{false} = KI \qquad \text{false}\ x\ y \to^* y$$

Church Booleans

Church encoding of boolean values *true* and *false*:

$$\text{true} = K \qquad \text{true}\ x\ y \to^* x$$
$$\text{false} = KI \qquad \text{false}\ x\ y \to^* y$$

Then we can define:

$$\text{or} = SII$$

$$\text{or}\ x\ y \to (Ix)(Ix)y \to^* xxy$$
$$\text{or true}\ y \to^* \text{true true}\ y \to^* \text{true}$$
$$\text{or false}\ y \to^* \text{false false}\ y \to^* y$$

Church Booleans

Church encoding of boolean values *true* and *false*:

$$\text{true} = K \qquad \text{true}\ x\ y \to^* x$$
$$\text{false} = KI \qquad \text{false}\ x\ y \to^* y$$

Then we can define:

$$\text{or} = SII$$

$$\text{or}\ x\ y \to (Ix)(Ix)y \to^* xxy$$
$$\text{or true}\ y \to^* \text{true true}\ y \to^* \text{true}$$
$$\text{or false}\ y \to^* \text{false false}\ y \to^* y$$

Exercise: define and and not.

Church Booleans

Church encoding of boolean values *true* and *false*:

$$\text{true} = K \qquad \text{true}\ x\ y \to^* x$$
$$\text{false} = KI \qquad \text{false}\ x\ y \to^* y$$

Then we can define:

$$\text{or} = SII$$

$$\text{or}\ x\ y \to (Ix)(Ix)y \to^* xxy$$
$$\text{or true}\ y \to^* \text{true true}\ y \to^* \text{true}$$
$$\text{or false}\ y \to^* \text{false false}\ y \to^* y$$

Exercise: define and and not.

If x then y else z:

$$\text{if} =$$

Church Booleans

Church encoding of boolean values *true* and *false*:

$$\text{true} = K \qquad \text{true}\ x\ y \to^* x$$
$$\text{false} = KI \qquad \text{false}\ x\ y \to^* y$$

Then we can define:

$$\text{or} = SII$$

$$\text{or}\ x\ y \to (Ix)(Ix)y \to^* xxy$$
$$\text{or true}\ y \to^* \text{true true}\ y \to^* \text{true}$$
$$\text{or false}\ y \to^* \text{false false}\ y \to^* y$$

Exercise: define and and not.

If x then y else z:

$$\text{if} = I$$

Church Booleans

Church encoding of boolean values *true* and *false*:

$$\text{true} = K \qquad \text{true}\ x\ y \to^* x$$
$$\text{false} = KI \qquad \text{false}\ x\ y \to^* y$$

Then we can define:

$$\text{or} = SII \qquad \text{or}\ x\ y \to (Ix)(Ix)y \to^* xxy$$
$$\text{or true}\ y \to^* \text{true true}\ y \to^* \text{true}$$
$$\text{or false}\ y \to^* \text{false false}\ y \to^* y$$

Exercise: define and and not.

If x then y else z:

$$\text{if} = I \qquad \text{if true}\ y\ z \to \text{true}\ y\ z \to^* y$$

Church Booleans

Church encoding of boolean values *true* and *false*:

$$\text{true} = K \qquad \text{true}\ x\ y \to^* x$$
$$\text{false} = KI \qquad \text{false}\ x\ y \to^* y$$

Then we can define:

$$\text{or} = SII$$
$$\text{or}\ x\ y \to (Ix)(Ix)y \to^* xxy$$
$$\text{or true}\ y \to^* \text{true true}\ y \to^* \text{true}$$
$$\text{or false}\ y \to^* \text{false false}\ y \to^* y$$

Exercise: define and and not.

If x then y else z:

$$\text{if} = I$$
$$\text{if true}\ y\ z \to \text{true}\ y\ z \to^* y$$
$$\text{if false}\ y\ z \to \text{false}\ y\ z \to^* z$$

Church Pairs

Church encoding of pairs: $\text{pair} = \lambda x.\ \lambda y.\ \lambda f.fxy$.

Church Pairs

Church encoding of pairs: $\text{pair} = \lambda x.\ \lambda y.\ \lambda f.fxy$. In CL:

$$\text{pair} = [xyf]\ f\ x\ y$$
$$\text{fst} = [p]\ p\ K$$
$$\text{snd} = [p]\ p\ (KI)$$

Church Pairs

Church encoding of pairs: $\text{pair} = \lambda x.\ \lambda y.\ \lambda f.fxy$. In CL:

$$\begin{aligned} \text{pair} &= [xyf]\ f\ x\ y \\ \text{fst} &= [p]\ p\ K \\ \text{snd} &= [p]\ p\ (KI) \end{aligned}$$

We have:

$$\text{pair}\ s\ t$$

Church Pairs

Church encoding of pairs: $\text{pair} = \lambda x.\ \lambda y.\ \lambda f.fxy$. In CL:

$$\begin{aligned} \text{pair} &= [xyf]\ f\ x\ y \\ \text{fst} &= [p]\ p\ K \\ \text{snd} &= [p]\ p\ (KI) \end{aligned}$$

We have:

$$\text{pair}\ s\ t = ([xyf]\ f\ x\ y)\ s\ t$$

Church Pairs

Church encoding of pairs: $\mathsf{pair} = \lambda x.\ \lambda y.\ \lambda f.fxy$. In CL:

$$\begin{aligned} \mathsf{pair} &= [xyf]\ f\ x\ y \\ \mathsf{fst} &= [p]\ p\ K \\ \mathsf{snd} &= [p]\ p\ (KI) \end{aligned}$$

We have:

$$\mathsf{pair}\ s\ t = ([xyf]\ f\ x\ y)\ s\ t \to^* ([f]\ f\ x\ y)[x \mapsto s,\ y \mapsto t]$$

Church Pairs

Church encoding of pairs: $\text{pair} = \lambda x.\ \lambda y.\ \lambda f.fxy$. In CL:

$$\begin{aligned} \text{pair} &= [xyf]\ f\ x\ y \\ \text{fst} &= [p]\ p\ K \\ \text{snd} &= [p]\ p\ (KI) \end{aligned}$$

We have:

$$\text{pair}\ s\ t = ([xyf]\ f\ x\ y)\ s\ t \to^* ([f]\ f\ x\ y)[x \mapsto s,\ y \mapsto t] \approx [f]\ f\ s\ t$$

Church Pairs

Church encoding of pairs: $\text{pair} = \lambda x.\ \lambda y.\ \lambda f.fxy$. In CL:

$$\begin{aligned} \text{pair} &= [xyf]\ f\ x\ y \\ \text{fst} &= [p]\ p\ K \\ \text{snd} &= [p]\ p\ (KI) \end{aligned}$$

We have:

$$\text{pair}\ s\ t = ([xyf]\ f\ x\ y)\ s\ t \to^* ([f]\ f\ x\ y)[x \mapsto s,\ y \mapsto t] \approx [f]\ f\ s\ t$$

$$\text{fst}\ (\text{pair}\ s\ t)$$

Church Pairs

Church encoding of pairs: $\text{pair} = \lambda x.\ \lambda y.\ \lambda f.fxy$. In CL:

$$\begin{aligned} \text{pair} &= [xyf]\ f\ x\ y \\ \text{fst} &= [p]\ p\ K \\ \text{snd} &= [p]\ p\ (KI) \end{aligned}$$

We have:

$$\text{pair}\ s\ t = ([xyf]\ f\ x\ y)\ s\ t \to^* ([f]\ f\ x\ y)[x \mapsto s,\ y \mapsto t] \approx [f]\ f\ s\ t$$

$$\text{fst}\ (\text{pair}\ s\ t) \to^* ([p]\ p\ K)\ ([f]\ f\ s\ t)$$

Church Pairs

Church encoding of pairs: $\text{pair} = \lambda x.\ \lambda y.\ \lambda f.fxy$. In CL:

$$\begin{aligned} \text{pair} &= [xyf]\ f\ x\ y \\ \text{fst} &= [p]\ p\ K \\ \text{snd} &= [p]\ p\ (KI) \end{aligned}$$

We have:

$$\text{pair}\ s\ t = ([xyf]\ f\ x\ y)\ s\ t \to^* ([f]\ f\ x\ y)[x \mapsto s,\ y \mapsto t] \approx [f]\ f\ s\ t$$

$$\text{fst}\ (\text{pair}\ s\ t) \to^* ([p]\ p\ K)\ ([f]\ f\ s\ t) \to^* ([f]\ f\ s\ t)K$$

Church Pairs

Church encoding of pairs: $\text{pair} = \lambda x.\ \lambda y.\ \lambda f.fxy$. In CL:

$$\begin{aligned} \text{pair} &= [xyf]\ f\ x\ y \\ \text{fst} &= [p]\ p\ K \\ \text{snd} &= [p]\ p\ (KI) \end{aligned}$$

We have:

$$\text{pair}\ s\ t = ([xyf]\ f\ x\ y)\ s\ t \to^* ([f]\ f\ x\ y)[x \mapsto s,\ y \mapsto t] \approx [f]\ f\ s\ t$$

$$\text{fst}\ (\text{pair}\ s\ t) \to^* ([p]\ p\ K)\ ([f]\ f\ s\ t) \to^* ([f]\ f\ s\ t)K \to^* K\ s\ t$$

Church Pairs

Church encoding of pairs: $\text{pair} = \lambda x.\ \lambda y.\ \lambda f.fxy$. In CL:

$$\begin{aligned} \text{pair} &= [xyf]\ f\ x\ y \\ \text{fst} &= [p]\ p\ K \\ \text{snd} &= [p]\ p\ (KI) \end{aligned}$$

We have:

$$\text{pair}\ s\ t = ([xyf]\ f\ x\ y)\ s\ t \to^* ([f]\ f\ x\ y)[x \mapsto s,\ y \mapsto t] \approx [f]\ f\ s\ t$$

$$\text{fst}\ (\text{pair}\ s\ t) \to^* ([p]\ p\ K)\ ([f]\ f\ s\ t) \to^* ([f]\ f\ s\ t)K \to^* K\ s\ t \to^* s$$

Church Pairs

Church encoding of pairs: $\text{pair} = \lambda x.\ \lambda y.\ \lambda f.fxy$. In CL:

$$\begin{aligned} \text{pair} &= [xyf]\ f\ x\ y \\ \text{fst} &= [p]\ p\ K \\ \text{snd} &= [p]\ p\ (KI) \end{aligned}$$

We have:

$$\begin{aligned} \text{pair}\ s\ t = ([xyf]\ f\ x\ y)\ s\ t \to^* ([f]\ f\ x\ y)[x \mapsto s,\ y \mapsto t] \approx [f]\ f\ s\ t \\ \text{fst}\ (\text{pair}\ s\ t) \to^* ([p]\ p\ K)\ ([f]\ f\ s\ t) \to^* ([f]\ f\ s\ t)K \to^* K\ s\ t \to^* s \\ \text{snd}\ (\text{pair}\ s\ t) \end{aligned}$$

Church Pairs

Church encoding of pairs: $\text{pair} = \lambda x.\ \lambda y.\ \lambda f.fxy$. In CL:

$$\begin{aligned} \text{pair} &= [xyf]\ f\ x\ y \\ \text{fst} &= [p]\ p\ K \\ \text{snd} &= [p]\ p\ (KI) \end{aligned}$$

We have:

$$\begin{aligned} \text{pair}\ s\ t = ([xyf]\ f\ x\ y)\ s\ t &\to^* ([f]\ f\ x\ y)[x \mapsto s,\ y \mapsto t] \approx [f]\ f\ s\ t \\ \text{fst}\ (\text{pair}\ s\ t) \to^* ([p]\ p\ K)\ ([f]\ f\ s\ t) &\to^* ([f]\ f\ s\ t)K \to^* K\ s\ t \to^* s \\ \text{snd}\ (\text{pair}\ s\ t) &\to^* ([p]\ p\ (KI))\ ([f]\ f\ s\ t) \end{aligned}$$

Church Pairs

Church encoding of pairs: $\text{pair} = \lambda x.\ \lambda y.\ \lambda f.fxy$. In CL:

$$\begin{aligned} \text{pair} &= [xyf]\ f\ x\ y \\ \text{fst} &= [p]\ p\ K \\ \text{snd} &= [p]\ p\ (KI) \end{aligned}$$

We have:

$$\begin{aligned} \text{pair}\ s\ t = ([xyf]\ f\ x\ y)\ s\ t &\to^* ([f]\ f\ x\ y)[x \mapsto s,\ y \mapsto t] \approx [f]\ f\ s\ t \\ \text{fst}\ (\text{pair}\ s\ t) \to^* ([p]\ p\ K)\ ([f]\ f\ s\ t) &\to^* ([f]\ f\ s\ t)K \to^* K\ s\ t \to^* s \\ \text{snd}\ (\text{pair}\ s\ t) \to^* ([p]\ p\ (KI))\ ([f]\ f\ s\ t) &\to^* ([f]\ f\ s\ t)(KI) \end{aligned}$$

Church Pairs

Church encoding of pairs: $\text{pair} = \lambda x.\ \lambda y.\ \lambda f.fxy$. In CL:

$$\begin{aligned} \text{pair} &= [xyf]\ f\ x\ y \\ \text{fst} &= [p]\ p\ K \\ \text{snd} &= [p]\ p\ (KI) \end{aligned}$$

We have:

$$\begin{aligned} \text{pair}\ s\ t = ([xyf]\ f\ x\ y)\ s\ t &\to^* ([f]\ f\ x\ y)[x \mapsto s,\ y \mapsto t] \approx [f]\ f\ s\ t \\ \text{fst}\ (\text{pair}\ s\ t) \to^* ([p]\ p\ K)\ ([f]\ f\ s\ t) &\to^* ([f]\ f\ s\ t)K \to^* K\ s\ t \to^* s \\ \text{snd}\ (\text{pair}\ s\ t) \to^* ([p]\ p\ (KI))\ ([f]\ f\ s\ t) &\to^* ([f]\ f\ s\ t)(KI) \to^* KI\ s\ t \end{aligned}$$

Church Pairs

Church encoding of pairs: $\text{pair} = \lambda x.\ \lambda y.\ \lambda f.fxy$. In CL:

$$\begin{aligned} \text{pair} &= [xyf]\ f\ x\ y \\ \text{fst} &= [p]\ p\ K \\ \text{snd} &= [p]\ p\ (KI) \end{aligned}$$

We have:

$$\begin{aligned} \text{pair}\ s\ t = ([xyf]\ f\ x\ y)\ s\ t &\to^* ([f]\ f\ x\ y)[x \mapsto s,\ y \mapsto t] \approx [f]\ f\ s\ t \\ \text{fst}\ (\text{pair}\ s\ t) \to^* ([p]\ p\ K)\ ([f]\ f\ s\ t) &\to^* ([f]\ f\ s\ t)K \to^* K\ s\ t \to^* s \\ \text{snd}\ (\text{pair}\ s\ t) \to^* ([p]\ p\ (KI))\ ([f]\ f\ s\ t) &\to^* ([f]\ f\ s\ t)(KI) \to^* KI\ s\ t \to^* t \end{aligned}$$

Church Pairs

Church encoding of pairs: $\text{pair} = \lambda x.\ \lambda y.\ \lambda f.fxy$. In CL:

$$\begin{aligned} \text{pair} &= [xyf]\ f\ x\ y \\ \text{fst} &= [p]\ p\ K \\ \text{snd} &= [p]\ p\ (KI) \end{aligned}$$

We have:

$$\begin{aligned} \text{pair}\ s\ t = ([xyf]\ f\ x\ y)\ s\ t &\to^* ([f]\ f\ x\ y)[x \mapsto s,\ y \mapsto t] \approx [f]\ f\ s\ t \\ \text{fst}\ (\text{pair}\ s\ t) \to^* ([p]\ p\ K)\ ([f]\ f\ s\ t) &\to^* ([f]\ f\ s\ t)K \to^* K\ s\ t \to^* s \\ \text{snd}\ (\text{pair}\ s\ t) \to^* ([p]\ p\ (KI))\ ([f]\ f\ s\ t) &\to^* ([f]\ f\ s\ t)(KI) \to^* KI\ s\ t \to^* t \end{aligned}$$

Exercise:

- compute $[xyf]\ f\ x\ y$, $[p]\ p\ K$, and $[p]\ p\ (KI)$.
- devise an encoding of triples

Church Numerals

Church encoding of natural numbers: $\overline{n} = \lambda f.\ \lambda x.\ f^n(x)$.

Church Numerals

Church encoding of natural numbers: $\overline{n} = \lambda f.\ \lambda x.\ f^n(x)$. In CL:

$$\overline{0} = KI$$
$$\overline{n+1} = ([nfx]\ f\ (n\ f\ x))\ \overline{n}$$

Church Numerals

Church encoding of natural numbers: $\overline{n} = \lambda f.\ \lambda x.\ f^n(x)$. In CL:

$$\overline{0} = KI$$

$$\overline{n+1} = ([nfx]\ f\ (n\ f\ x))\ \overline{n} \approx S\ (S(KS)(S(KK)I))\ \overline{n}$$

Church Numerals

Church encoding of natural numbers: $\overline{n} = \lambda f.\ \lambda x.\ f^n(x)$. In CL:

$$\overline{0} = KI$$
$$\overline{n+1} = ([nfx]\ f\ (n\ f\ x))\ \overline{n} \approx S\ (S(KS)(S(KK)I))\ \overline{n}$$

$\overline{0}\ f\ x \to^* x$

Church Numerals

Church encoding of natural numbers: $\overline{n} = \lambda f.\ \lambda x.\ f^n(x)$. In CL:

$$\overline{0} = KI$$
$$\overline{n+1} = ([nfx]\ f\ (n\ f\ x))\ \overline{n} \approx S\ (S(KS)(S(KK)I))\ \overline{n}$$

$\overline{0}\ f\ x \to^* x$ $\qquad\qquad$ $\overline{n+1}\ f\ x$

Church Numerals

Church encoding of natural numbers: $\overline{n} = \lambda f.\ \lambda x.\ f^n(x)$. In CL:

$$\overline{0} = KI$$
$$\overline{n+1} = ([nfx]\ f\ (n\ f\ x))\ \overline{n} \approx S\ (S(KS)(S(KK)I))\ \overline{n}$$

$\overline{0}\ f\ x \to^* x$ $\qquad$ $\overline{n+1}\ f\ x = S\ (S(KS)(S(KK)I))\ \overline{n}\ f\ x$

Church Numerals

Church encoding of natural numbers: $\overline{n} = \lambda f.\ \lambda x.\ f^n(x)$. In CL:

$$\overline{0} = KI$$
$$\overline{n+1} = ([nfx]\ f\ (n\ f\ x))\ \overline{n} \approx S\ (S(KS)(S(KK)I))\ \overline{n}$$

$\overline{0}\ f\ x \to^* x$

$$\overline{n+1}\ f\ x = S\ (S(KS)(S(KK)I))\ \overline{n}\ f\ x$$
$$\to S(KS)(S(KK)I)\ f\ (\overline{n}\ f)\ x$$

Church Numerals

Church encoding of natural numbers: $\overline{n} = \lambda f.\ \lambda x.\ f^n(x)$. In CL:

$$\overline{0} = KI$$
$$\overline{n+1} = ([nfx]\ f\ (n\ f\ x))\ \overline{n} \approx S\ (S(KS)(S(KK)I))\ \overline{n}$$

$\overline{0}\ f\ x \to^* x$

$$\begin{aligned}\overline{n+1}\ f\ x &= S\ (S(KS)(S(KK)I))\ \overline{n}\ f\ x \\ &\to S(KS)(S(KK)I)\ f\ (\overline{n}\ f)\ x \\ &\to KSf(S(KK)I\ f)\ (\overline{n}\ f)\ x\end{aligned}$$

Church Numerals

Church encoding of natural numbers: $\overline{n} = \lambda f.\ \lambda x.\ f^n(x)$. In CL:

$$\overline{0} = KI$$
$$\overline{n+1} = ([nfx]\ f\ (n\ f\ x))\ \overline{n} \approx S\ (S(KS)(S(KK)I))\ \overline{n}$$

$\overline{0}\ f\ x \to^* x$

$$\begin{aligned}\overline{n+1}\ f\ x &= S\ (S(KS)(S(KK)I))\ \overline{n}\ f\ x \\ &\to S(KS)(S(KK)I)\ f\ (\overline{n}\ f)\ x \\ &\to KSf(S(KK)I\ f)\ (\overline{n}\ f)\ x \\ &\to S(S(KK)If)\ (\overline{n}\ f)\ x\end{aligned}$$

Church Numerals

Church encoding of natural numbers: $\overline{n} = \lambda f.\ \lambda x.\ f^n(x)$. In CL:

$$\overline{0} = KI$$
$$\overline{n+1} = ([nfx]\ f\ (n\ f\ x))\ \overline{n} \approx S\ (S(KS)(S(KK)I))\ \overline{n}$$

$\overline{0}\ f\ x \to^* x$

$$\begin{aligned}
\overline{n+1}\ f\ x &= S\ (S(KS)(S(KK)I))\ \overline{n}\ f\ x \\
&\to S(KS)(S(KK)I)\ f\ (\overline{n}\ f)\ x \\
&\to KSf(S(KK)I\ f)\ (\overline{n}\ f)\ x \\
&\to S(S(KK)If)\ (\overline{n}\ f)\ x \\
&\to S(KKf(I\ f))\ (\overline{n}\ f)\ x
\end{aligned}$$

Church Numerals

Church encoding of natural numbers: $\overline{n} = \lambda f.\ \lambda x.\ f^n(x)$. In CL:

$$\overline{0} = KI$$
$$\overline{n+1} = ([nfx]\ f\ (n\ f\ x))\ \overline{n} \approx S\ (S(KS)(S(KK)I))\ \overline{n}$$

$\overline{0}\ f\ x \to^* x$

$$\begin{aligned}
\overline{n+1}\ f\ x &= S\ (S(KS)(S(KK)I))\ \overline{n}\ f\ x \\
&\to S(KS)(S(KK)I)\ f\ (\overline{n}\ f)\ x \\
&\to KSf(S(KK)I\ f)\ (\overline{n}\ f)\ x \\
&\to S(S(KK)If)\ (\overline{n}\ f)\ x \\
&\to S(KKf(I\ f))\ (\overline{n}\ f)\ x \\
&\to S(K(I\ f))\ (\overline{n}\ f)\ x
\end{aligned}$$

Church Numerals

Church encoding of natural numbers: $\overline{n} = \lambda f.\ \lambda x.\ f^n(x)$. In CL:

$$\overline{0} = KI$$
$$\overline{n+1} = ([nfx]\ f\ (n\ f\ x))\ \overline{n} \approx S\ (S(KS)(S(KK)I))\ \overline{n}$$

$\overline{0}\ f\ x \to^* x$

$$\begin{aligned}
\overline{n+1}\ f\ x &= S\ (S(KS)(S(KK)I))\ \overline{n}\ f\ x \\
&\to S(KS)(S(KK)I)\ f\ (\overline{n}\ f)\ x \\
&\to KSf(S(KK)I\ f)\ (\overline{n}\ f)\ x \\
&\to S(S(KK)If)\ (\overline{n}\ f)\ x \\
&\to S(KKf(I\ f))\ (\overline{n}\ f)\ x \\
&\to S(K(I\ f))\ (\overline{n}\ f)\ x \\
&\to S(K\ f)\ (\overline{n}\ f)\ x
\end{aligned}$$

Church Numerals

Church encoding of natural numbers: $\overline{n} = \lambda f.\ \lambda x.\ f^n(x)$. In CL:

$$\overline{0} = KI$$
$$\overline{n+1} = ([nfx]\ f\ (n\ f\ x))\ \overline{n} \approx S\ (S(KS)(S(KK)I))\ \overline{n}$$

$\overline{0}\ f\ x \to^* x$

$$\begin{aligned}
\overline{n+1}\ f\ x &= S\ (S(KS)(S(KK)I))\ \overline{n}\ f\ x \\
&\to S(KS)(S(KK)I)\ f\ (\overline{n}\ f)\ x \\
&\to KSf(S(KK)I\ f)\ (\overline{n}\ f)\ x \\
&\to S(S(KK)If)\ (\overline{n}\ f)\ x \\
&\to S(KKf(I\ f))\ (\overline{n}\ f)\ x \\
&\to S(K(I\ f))\ (\overline{n}\ f)\ x \\
&\to S(K\ f)\ (\overline{n}\ f)\ x \\
&\to Kfx\ (\overline{n}\ f\ x)
\end{aligned}$$

Church Numerals

Church encoding of natural numbers: $\overline{n} = \lambda f.\ \lambda x.\ f^n(x)$. In CL:

$$\overline{0} = KI$$
$$\overline{n+1} = ([nfx]\ f\ (n\ f\ x))\ \overline{n} \approx S\ (S(KS)(S(KK)I))\ \overline{n}$$

$\overline{0}\ f\ x \to^* x$

$$\begin{aligned}
\overline{n+1}\ f\ x &= S\ (S(KS)(S(KK)I))\ \overline{n}\ f\ x \\
&\to S(KS)(S(KK)I)\ f\ (\overline{n}\ f)\ x \\
&\to KSf(S(KK)I\ f)\ (\overline{n}\ f)\ x \\
&\to S(S(KK)If)\ (\overline{n}\ f)\ x \\
&\to S(KKf(I\ f))\ (\overline{n}\ f)\ x \\
&\to S(K(I\ f))\ (\overline{n}\ f)\ x \\
&\to S(K\ f)\ (\overline{n}\ f)\ x \\
&\to Kfx\ (\overline{n}\ f\ x) \\
&\to f(\overline{n}\ f\ x)
\end{aligned}$$

Church Numerals

Church encoding of natural numbers: $\overline{n} = \lambda f.\ \lambda x.\ f^n(x)$. In CL:

$$\overline{0} = KI$$
$$\overline{n+1} = ([nfx]\ f\ (n\ f\ x))\ \overline{n} \approx S\ (S(KS)(S(KK)I))\ \overline{n}$$

$\overline{0}\ f\ x \to^* x$

$$\begin{aligned}
\overline{n+1}\ f\ x &= S\ (S(KS)(S(KK)I))\ \overline{n}\ f\ x \\
&\to S(KS)(S(KK)I)\ f\ (\overline{n}\ f)\ x \\
&\to KSf(S(KK)I\ f)\ (\overline{n}\ f)\ x \\
&\to S(S(KK)If)\ (\overline{n}\ f)\ x \\
&\to S(KKf(I\ f))\ (\overline{n}\ f)\ x \\
&\to S(K(I\ f))\ (\overline{n}\ f)\ x \\
&\to S(K\ f)\ (\overline{n}\ f)\ x \\
&\to Kfx\ (\overline{n}\ f\ x) \\
&\to f(\overline{n}\ f\ x) \\
&\to^* f^{n+1}(x)
\end{aligned}$$

Computation with Church Numerals

- $\mathsf{plus} = [m\, n\, f\, x]\ m\, f\, (n\, f\, x)$

Computation with Church Numerals

- plus $= [m\, n\, f\, x]\ m\, f\, (n\, f\, x)$
- succ $= [n\, f\, x]\ f\, (n\, f\, x)$

Computation with Church Numerals

- plus $= [m\, n\, f\, x]\ m\, f\, (n\, f\, x)$
- succ $= [n\, f\, x]\ f\, (n\, f\, x)$
- isZero $= [n]\ n\, (K\ \text{false})\ \text{true}$

Computation with Church Numerals

- plus $= [m\,n\,f\,x]\ m\,f\,(n\,f\,x)$
- succ $= [n\,f\,x]\ f\,(n\,f\,x)$
- isZero $= [n]\ n\,(K\ \text{false})\ \text{true}$
- pred $= [n\,f\,x]\ n\,([g\,h]\ h\,(g\,f))\,(Kx)\ I$

Computation with Church Numerals

- plus $= [m\, n\, f\, x]\ m\, f\, (n\, f\, x)$
- succ $= [n\, f\, x]\ f\, (n\, f\, x)$
- isZero $= [n]\ n\, (K\ \text{false})\ \text{true}$
- pred $= [n\, f\, x]\ n\, ([g\, h]\ h\, (g\, f))\ (K x)\ I$
- ...

Idea behind predecessor:

$$\text{pred} = \lambda n.\ \text{fst}\ (n\ f\ z) \qquad z = (0, 0)$$
$$f\ (x, y) = (y, y + 1)$$

E.g. pred $\overline{3} = \text{fst}\ f(f(f(0,0))) = \text{fst}\ f(f(0,1)) = \text{fst}\ f(1,2) = \text{fst}\ (2,3) = 2.$

Computable Functions

Computable functions $\approx$ everything a computer with infinite memory can compute.

Computable Functions

Computable functions $\approx$ everything a computer with infinite memory can compute.

The class of computable functions can be defined using different models:

- Turing machines
- Lambda calculus
- Post machines
- Register machines
- μ-recursive functions

Computable Functions in Combinatory Logic

The class of μ-recursive (found by Kleene) functions is build from:

- *zero*: 0

Computable Functions in Combinatory Logic

The class of μ-recursive (found by Kleene) functions is build from:

- *zero*: 0 in CL: KI

Computable Functions in Combinatory Logic

The class of μ-recursive (found by Kleene) functions is build from:

- *zero*: 0 — in CL: KI
- *successor*: S(n) = n+1

Computable Functions in Combinatory Logic

The class of μ-recursive (found by Kleene) functions is build from:

- *zero*: 0 — in CL: KI
- *successor*: S(n) = n+1 — in CL: $[n\,f\,x]\ f\ (n\,f\,x)$

Computable Functions in Combinatory Logic

The class of μ-recursive (found by Kleene) functions is build from:

- *zero*: 0 — in CL: KI
- *successor*: S(n) = n+1 — in CL: $[n\,f\,x]\ f\ (n\,f\,x)$
- *projection functions*: $\Pi^i_k(n_1, \ldots, n_k) = n_i$

Computable Functions in Combinatory Logic

The class of μ-recursive (found by Kleene) functions is build from:

- *zero*: 0 — in CL: KI
- *successor*: $\mathsf{S}(\mathsf{n}) = \mathsf{n}{+}1$ — in CL: $[n\,f\,x]\ f\ (n\,f\,x)$
- *projection functions*: $\Pi^i_k(n_1, \ldots, n_k) = n_i$ — in CL: pair, fst, snd, ...

Computable Functions in Combinatory Logic

The class of μ-recursive (found by Kleene) functions is build from:

- *zero*: 0 — in CL: KI
- *successor*: S(n) = n+1 — in CL: $[n\,f\,x]\ f\ (n\,f\,x)$
- *projection functions*: $\Pi^i_k(n_1, \ldots, n_k) = n_i$ — in CL: pair, fst, snd, ...
- *composition*: $f(x_1, \ldots, x_n) = g(h_1(\vec{x}), \ldots, h_m(\vec{x}))$

Computable Functions in Combinatory Logic

The class of μ-recursive (found by Kleene) functions is build from:

- *zero*: 0 — in CL: KI
- *successor*: $\mathsf{S}(\mathsf{n}) = \mathsf{n}{+}1$ — in CL: $[n\,f\,x]\ f\ (n\,f\,x)$
- *projection functions*: $\Pi^i_k(n_1, \ldots, n_k) = n_i$ — in CL: pair, fst, snd, $\ldots$
- *composition*: $f(x_1, \ldots, x_n) = g(h_1(\vec{x}), \ldots, h_m(\vec{x}))$ — in CL: $[\vec{x}]g(h_1\ \vec{x}, \ldots, h_m\ \vec{x})$

Computable Functions in Combinatory Logic

The class of μ-recursive (found by Kleene) functions is build from:

- *zero*: 0 — in CL: KI
- *successor*: $\mathsf{S(n)} = \mathsf{n{+}1}$ — in CL: $[n\,f\,x]\ f\ (n\,f\,x)$
- *projection functions*: $\Pi^i_k(n_1, \ldots, n_k) = n_i$ — in CL: pair, fst, snd, ...
- *composition*: $f(x_1, \ldots, x_n) = g(h_1(\vec{x}), \ldots, h_m(\vec{x}))$
  in CL: $[\vec{x}]g(h_1\ \vec{x}, \ldots, h_m\ \vec{x})$
- *primitive recursion*: $f(0, \vec{x}) = g(\vec{x})$
  $f(S(n), \vec{x}) = h(f(n, \vec{x}), n, \vec{x})$

Computable Functions in Combinatory Logic

The class of μ-recursive (found by Kleene) functions is build from:

- *zero*: 0 — in CL: KI
- *successor*: S(n) = n+1 — in CL: $[n\,f\,x]\ f\ (n\,f\,x)$
- *projection functions*: $\Pi^i_k(n_1, \ldots, n_k) = n_i$ — in CL: pair, fst, snd, ...
- *composition*: $f(x_1, \ldots, x_n) = g(h_1(\vec{x}), \ldots, h_m(\vec{x}))$
  in CL: $[\vec{x}]g(h_1\ \vec{x}, \ldots, h_m\ \vec{x})$
- *primitive recursion*: $f(0, \vec{x}) = g(\vec{x})$
  $f(S(n), \vec{x}) = h(f(n, \vec{x}), n, \vec{x})$
  in CL: implicit definition $f\ n\ \vec{x}\ =$ isZero $n\ (g\ \vec{x})\ (h\ (f\ (\text{pred}\ n)\ \vec{x})\ n\ \vec{x})$

Computable Functions in Combinatory Logic

The class of μ-recursive (found by Kleene) functions is build from:

- *zero*: 0 — in CL: KI
- *successor*: S(n) = n+1 — in CL: $[n\,f\,x]\ f\ (n\,f\,x)$
- *projection functions*: $\Pi^i_k(n_1, \ldots, n_k) = n_i$ — in CL: pair, fst, snd, ...
- *composition*: $f(x_1, \ldots, x_n) = g(h_1(\vec{x}), \ldots, h_m(\vec{x}))$
  in CL: $[\vec{x}]g(h_1\ \vec{x}, \ldots, h_m\ \vec{x})$
- *primitive recursion*: $f(0, \vec{x}) = g(\vec{x})$
  $f(S(n), \vec{x}) = h(f(n, \vec{x}), n, \vec{x})$
  in CL: implicit definition $f\ n\ \vec{x}\ =$ isZero $n\ (g\ \vec{x})\ (h\ (f\ (\text{pred}\ n)\ \vec{x})\ n\ \vec{x})$
- *unbounded search*: $\mu u.[f(u, \vec{x}) = 0]$ is the least u such that $f(u, \vec{x}) = 0$

Computable Functions in Combinatory Logic

The class of μ-recursive (found by Kleene) functions is build from:

- *zero*: 0 — in CL: KI
- *successor*: S(n) = n+1 — in CL: $[n\,f\,x]\ f\ (n\,f\,x)$
- *projection functions*: $\Pi^i_k(n_1, \ldots, n_k) = n_i$ — in CL: pair, fst, snd, ...
- *composition*: $f(x_1, \ldots, x_n) = g(h_1(\vec{x}), \ldots, h_m(\vec{x}))$
  in CL: $[\vec{x}]g(h_1\ \vec{x}, \ldots, h_m\ \vec{x})$
- *primitive recursion*: $f(0, \vec{x}) = g(\vec{x})$
  $f(S(n), \vec{x}) = h(f(n, \vec{x}), n, \vec{x})$
  in CL: implicit definition $f\ n\ \vec{x}\ =$ isZero $n\ (g\ \vec{x})\ (h\ (f\ (\text{pred}\ n)\ \vec{x})\ n\ \vec{x})$
- *unbounded search*: $\mu u.[f(u, \vec{x}) = 0]$ is the least u such that $f(u, \vec{x}) = 0$
  in CL: implicit definition $\mu\ u\ f\ \vec{x}\ =$ isZero $(f\ u\ \vec{x})\ u\ (\mu\ (\text{succ}\ u)\ f\ \vec{x})$

Computable Functions in Combinatory Logic

The class of μ-recursive (found by Kleene) functions is build from:

- *zero*: 0 — in CL: KI
- *successor*: S(n) = n+1 — in CL: $[n\,f\,x]\ f\ (n\,f\,x)$
- *projection functions*: $\Pi^i_k(n_1, \ldots, n_k) = n_i$ — in CL: pair, fst, snd, ...
- *composition*: $f(x_1, \ldots, x_n) = g(h_1(\vec{x}), \ldots, h_m(\vec{x}))$
  in CL: $[\vec{x}]g(h_1\ \vec{x}, \ldots, h_m\ \vec{x})$
- *primitive recursion*: $f(0, \vec{x}) = g(\vec{x})$
  $f(S(n), \vec{x}) = h(f(n, \vec{x}), n, \vec{x})$
  in CL: implicit definition $f\ n\ \vec{x}\ =\ \text{isZero}\ n\ (g\ \vec{x})\ (h\ (f\ (\text{pred}\ n)\ \vec{x})\ n\ \vec{x})$
- *unbounded search*: $\mu u.[f(u, \vec{x}) = 0]$ is the least u such that $f(u, \vec{x}) = 0$
  in CL: implicit definition $\mu\ u\ f\ \vec{x}\ =\ \text{isZero}\ (f\ u\ \vec{x})\ u\ (\mu\ (\text{succ}\ u)\ f\ \vec{x})$

Every computable function can be computed in combinatory logic.

Currying

Example

Currying $A(x, S(y))$ gives $A\ x\ (Sy)$

Currying

Example

Currying $A(x, S(y))$ gives $A\ x\ (Sy)$

- One binary function symbol Ap and for the rest only constants.

Currying

Example

Currying $A(x, S(y))$ gives $A\ x\ (Sy)$

- One binary function symbol Ap and for the rest only constants.

Definition

For each TRS (Σ, R) we define a *curried* version $(\Sigma, R)^{cur} = (\Sigma^{cur},\ R^{cur})$.

R^{cur} has rules $cur(t) \to cur(s)$ for $t \to s$ in R, where:

$$\begin{array}{lcl} cur(x) & = & x \\ cur(F(t_1, \ldots, t_n)) & = & F\ cur(t_1)\ \ldots\ cur(t_n) \end{array}$$