

- 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)

$$Ap(Ap(Ap(S, x), y), z) \rightarrow Ap(Ap(x, z), Ap(y, z))$$

$$Ap(Ap(K, x), y) \rightarrow x$$

$$Ap(I, x) \rightarrow x$$

# 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)

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

## CL in infix notation

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

## CL in standard notation

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

# Association to the Left

## Association to the Left

A term  $t_1 t_2 t_3 \dots t_n$  reduces to  $((\dots ((t_1 t_2) t_3) \dots) t_n)$

# Association to the Left

## Association to the Left

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

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



# Association to the Left

## Association to the Left

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

- $xz(yz)$  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  $t_1 t_2 t_3 \dots t_n$  restores to  $((\dots((t_1 t_2)t_3)\dots)t_n)$

- $xz(yz)$  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  $t_1 t_2 t_3 \dots t_n$  restores to  $((\dots((t_1 t_2)t_3)\dots)t_n)$

- $xz(yz)$  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  $S!x$  does not contain a redex  $!x$ .

# 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:

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

# A Famous Term

- A famous term with a famous reduction cycle:

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

# A Famous Term

- A famous term with a famous reduction cycle:

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

- 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:

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

- Let  $D = SII$ .

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

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

# A Famous Term

- A famous term with a famous reduction cycle:

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

- Let  $D = SII$ .

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

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

# A Famous Term

- A famous term with a famous reduction cycle:

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

- Let  $D = SII$ .

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

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

# 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

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

# Combinators

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

We have

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

# Combinators

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

We have

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



# Combinators

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

We have

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

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

We have

$$Cxyz \rightarrow^* xzy$$

# Combinators

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

We have

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

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

We have

$$Cxyz \rightarrow^* xzy$$

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

# Combinatorial Completeness

## Lemma (Combinatorial Completeness)

*Given CL-term  $t$ , one can find a CL-term  $F$  such that*

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

*This  $F$  can be constructed such that the the variables  $x_1, \dots, x_n$  do not occur in  $F$ .*

# Combinatorial Completeness

## Lemma (Combinatorial Completeness)

Given CL-term  $t$ , one can find a CL-term  $F$  such that

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

This  $F$  can be constructed such that the variables  $x_1, \dots, x_n$  do not occur in  $F$ .

## Lemma

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

# Combinatorial Completeness

## Lemma (Combinatorial Completeness)

Given CL-term  $t$ , one can find a CL-term  $F$  such that

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

This  $F$  can be constructed such that the variables  $x_1, \dots, x_n$  do not occur in  $F$ .

Then closure under substitutions yields:

## Lemma

Then  $F t_1 \dots t_n \rightarrow^* t[x_1 \mapsto t_1, \dots, x_n \mapsto t_n]$  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  $[x]tt' = S([x]t)([x]t')$ .

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

$$\begin{aligned} \blacksquare \quad t' &= [x]t = [x](SI(Kx)) = S([x](SI))([x](Kx)) \\ &= S(K(SI))(S([x]K)([x]x)) = S(K(SI))(S(KK)I). \end{aligned}$$

## 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  $[x_1]([x_2](\dots([x_n]t)\dots))$  we will write  $[x_1x_2\dots x_n]t$

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

$$\begin{aligned} \blacksquare \quad t' &= [x]t = [x](SI(Kx)) = S([x](SI))([x](Kx)) \\ &= S(K(SI))(S([x]K)([x]x)) = S(K(SI))(S(KK)I). \end{aligned}$$

# 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  $[x_1]([x_2](\dots([x_n]t)\dots))$  we will write  $[x_1x_2\dots x_n]t$

## 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  $[x_1]([x_2](\dots([x_n]t)\dots))$  we will write  $[x_1x_2\dots x_n]t$

## 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  $[x_1]([x_2](\dots([x_n]t)\dots))$  we will write  $[x_1x_2\dots x_n]t$

## 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  $[x_1]([x_2](\dots([x_n]t)\dots))$  we will write  $[x_1x_2\dots x_n]t$

## 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  $[x_1]([x_2](\dots([x_n]t)\dots))$  we will write  $[x_1x_2\dots x_n]t$

## 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  $[x_1]([x_2](\dots([x_n]t)\dots))$  we will write  $[x_1x_2\dots x_n]t$

## 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  $[x_1]([x_2](\dots([x_n]t)\dots))$  we will write  $[x_1x_2\dots x_n]t$

## 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  $[x_1]([x_2](\dots([x_n]t)\dots))$  we will write  $[x_1x_2\dots x_n]t$

## 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  $[x_1]([x_2](\dots([x_n]t)\dots))$  we will write  $[x_1x_2\dots x_n]t$

## 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**  $([x]t)x \rightarrow^* t$

**2** *The variable  $x$  does not occur in the CL-term denoted by  $[x]t$*

## Towards a Proof of Combinatorial Completeness

## Lemma (Properties of )

- 1  $([x]t)x \rightarrow^* t$

- 2 *The variable  $x$  does not occur in the CL-term denoted by  $[x]t$*

## Proof.

Induction on  $t$ :

base case  $([x]x)x = lx \rightarrow x$

$$([x]y)x = Kyx \rightarrow y \text{ (the same if } t \text{ is a constant)}$$

## Towards a Proof of Combinatorial Completeness

## Lemma (Properties of )

- 1  $([x]t)x \rightarrow^* t$

- 2 *The variable  $x$  does not occur in the CL-term denoted by  $[x]t$*

## Proof.

Induction on  $t$ :

base case  $([x]x)x = lx \rightarrow x$

$$([x]y)x = Kyx \rightarrow y \text{ (the same if } t \text{ is a constant)}$$

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

## Towards a Proof of Combinatorial Completeness

## Lemma (Properties of )

- 1  $([x]t)x \rightarrow^* t$

- 2 *The variable  $x$  does not occur in the CL-term denoted by  $[x]t$*

## Proof.

Induction on  $t$ :

base case  $([x]x)x = lx \rightarrow x$

$$([x]y)x = Kyx \rightarrow y \text{ (the same if } t \text{ is a constant)}$$

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

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



# Simulation of beta reduction

## Lemma

*We can use abstraction to simulate  $\beta$ -reduction of  $\lambda$ -calculus:*

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

# Simulation of beta reduction

## Lemma

*We can use abstraction to simulate  $\beta$ -reduction of  $\lambda$ -calculus:*

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

## Proof.

We have  $([x]t)x \rightarrow^* t$ .

Substitute  $t'$  for  $x$  in this reduction. ■

# Proof of Combinatorial Completeness

## Combinatorial Completeness

Given a CL-term  $t$ , find a CL-term  $F$  such that  $Fx_1 \dots x_n \rightarrow^* t$

By former proposition and induction on  $n$ :

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



# Proof of Combinatorial Completeness

## Combinatorial Completeness

Given a CL-term  $t$ , find a CL-term  $F$  such that  $Fx_1 \dots x_n \rightarrow^* t$

## Proof.

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



# Proof of Combinatorial Completeness

## Combinatorial Completeness

Given a CL-term  $t$ , find a CL-term  $F$  such that  $Fx_1 \dots x_n \rightarrow^* t$

### Proof.

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

By former proposition and induction on  $n$ :

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



# 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:

$$P_F = D(BFD)$$

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

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:

$$P_F = D(BFD)$$

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

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:

$$P_F = D(BFD)$$

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

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:

$$P_F = D(BFD)$$

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

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

$$F x_1 \dots x_n \leftrightarrow^* t[y := F]$$

# Implicit function definition

Given a CL-term  $t$ , find  $F$  such that

$$F x_1 \dots x_n \leftrightarrow^* t[y := F]$$

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

$$F x_1 \dots x_n \leftrightarrow^* t[y := F]$$

We take:

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

Then:

$$F x_1 \dots x_n = t' F x_1 \dots x_n = t' y x_1 \dots x_n [y := F] \rightarrow^* t[y := F]$$

# Implicit function definition

Given a CL-term  $t$ , find  $F$  such that

$$F x_1 \dots x_n \leftrightarrow^* t[y := F]$$

We take:

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

Then:

$$F x_1 \dots x_n = t' F x_1 \dots x_n = t' y x_1 \dots x_n [y := F] \rightarrow^* t[y := F]$$

Application: **recursion**.



# 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  $A \times (Sy)$

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

## Definition

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

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

$$\begin{aligned} cur(x) &= x \\ cur(F(t_1, \dots, t_n)) &= F cur(t_1) \dots cur(t_n) \end{aligned}$$

# Church Booleans

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

$$\text{true} = K$$

# Church Booleans

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

$$\text{true} = K$$

$$\text{false} = KI$$

$$\text{true } x y \rightarrow^* x$$

# Church Booleans

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

$$\text{true} = K$$

$$\text{false} = KI$$

$$\text{true } x y \rightarrow^* x$$

$$\text{false } x y \rightarrow^* y$$

# Church Booleans

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

$$\text{true} = K$$

$$\text{true } x y \rightarrow^* x$$

$$\text{false} = KI$$

$$\text{false } x y \rightarrow^* y$$

Then we can define:

$$\text{or} = SII$$

$$\text{or } x y \rightarrow (Ix)(Ix)y \rightarrow^* xxy$$

$$\text{or true } y \rightarrow^* \text{true true } y \rightarrow^* \text{true}$$

$$\text{or false } y \rightarrow^* \text{false false } y \rightarrow^* y$$

# Church Booleans

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

$$\text{true} = K$$

$$\text{false} = KI$$

$$\text{true } x y \rightarrow^* x$$

$$\text{false } x y \rightarrow^* y$$

Then we can define:

$$\text{or} = SII$$

$$\text{or } x y \rightarrow (Ix)(Ix)y \rightarrow^* xxy$$

$$\text{or true } y \rightarrow^* \text{true true } y \rightarrow^* \text{true}$$

$$\text{or false } y \rightarrow^* \text{false false } y \rightarrow^* y$$

Exercise: define **and** and **not**.



# Church Booleans

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

$$\text{true} = K$$

$$\text{true } x y \rightarrow^* x$$

$$\text{false} = KI$$

$$\text{false } x y \rightarrow^* y$$

Then we can define:

$$\text{or} = SII$$

$$\text{or } x y \rightarrow (Ix)(Ix)y \rightarrow^* xxy$$

$$\text{or true } y \rightarrow^* \text{true true } y \rightarrow^* \text{true}$$

$$\text{or false } y \rightarrow^* \text{false false } y \rightarrow^* y$$

Exercise: define **and** and **not**.

If *x* then *y* else *z*:

$$\text{if} =$$

# Church Booleans

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

$$\text{true} = K$$

$$\text{true } x y \rightarrow^* x$$

$$\text{false} = KI$$

$$\text{false } x y \rightarrow^* y$$

Then we can define:

$$\text{or} = SII$$

$$\text{or } x y \rightarrow (Ix)(Ix)y \rightarrow^* xxy$$

$$\text{or true } y \rightarrow^* \text{true true } y \rightarrow^* \text{true}$$

$$\text{or false } y \rightarrow^* \text{false false } y \rightarrow^* y$$

Exercise: define **and** and **not**.

If *x* then *y* else *z*:

$$\text{if} = I$$

# Church Booleans

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

$$\text{true} = K$$

$$\text{true } x y \rightarrow^* x$$

$$\text{false} = KI$$

$$\text{false } x y \rightarrow^* y$$

Then we can define:

$$\text{or} = SII$$

$$\text{or } x y \rightarrow (Ix)(Ix)y \rightarrow^* xxy$$

$$\text{or true } y \rightarrow^* \text{true true } y \rightarrow^* \text{true}$$

$$\text{or false } y \rightarrow^* \text{false false } y \rightarrow^* y$$

Exercise: define **and** and **not**.

If *x* then *y* else *z*:

$$\text{if} = I$$

$$\text{if true } y z \rightarrow \text{true } y z \rightarrow^* y$$

# Church Booleans

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

$$\text{true} = K$$

$$\text{true } x y \rightarrow^* x$$

$$\text{false} = KI$$

$$\text{false } x y \rightarrow^* y$$

Then we can define:

$$\text{or} = SII$$

$$\text{or } x y \rightarrow (Ix)(Ix)y \rightarrow^* xxy$$

$$\text{or true } y \rightarrow^* \text{true true } y \rightarrow^* \text{true}$$

$$\text{or false } y \rightarrow^* \text{false false } y \rightarrow^* y$$

Exercise: define **and** and **not**.

If *x* then *y* else *z*:

$$\text{if} = I$$

$$\text{if true } y z \rightarrow \text{true } y z \rightarrow^* y$$

$$\text{if false } y z \rightarrow \text{false } y z \rightarrow^* z$$

# 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:

$$\text{pair} = [xyf] f x y$$

$$\text{fst} = [\rho] \rho K$$

$$\text{snd} = [\rho] \rho (KI)$$

# Church Pairs

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

$$\text{pair} = [xyf] f x y$$

$$\text{fst} = [\rho] \rho K$$

$$\text{snd} = [\rho] \rho (KI)$$

We have:

$\text{pair } s \ t$

# Church Pairs

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

$$\text{pair} = [xyf] f x y$$

$$\text{fst} = [\rho] \rho K$$

$$\text{snd} = [\rho] \rho (KI)$$

We have:

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



# Church Pairs

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

$$\text{pair} = [xyf] f x y$$

$$\text{fst} = [\rho] \rho K$$

$$\text{snd} = [\rho] \rho (KI)$$

We have:

$$\text{pair } s \ t = [xyf] f x y \rightarrow^* [f] f s t$$

# Church Pairs

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

$$\text{pair} = [xyf] f x y$$

$$\text{fst} = [\rho] \rho K$$

$$\text{snd} = [\rho] \rho (KI)$$

We have:

$$\begin{aligned} \text{pair } s \ t &= [xyf] f x y \rightarrow^* [f] f s t \\ \text{fst } (\text{pair } s \ t) & \end{aligned}$$

# Church Pairs

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

$$\text{pair} = [xyf] f x y$$

$$\text{fst} = [\rho] \rho K$$

$$\text{snd} = [\rho] \rho (KI)$$

We have:

$$\text{pair } s \ t = [xyf] f x y \rightarrow^* [f] f s t$$

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

# Church Pairs

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

$$\text{pair} = [xyf] f x y$$

$$\text{fst} = [\rho] \rho K$$

$$\text{snd} = [\rho] \rho (KI)$$

We have:

$$\text{pair } s t = [xyf] f x y \rightarrow^* [f] f s t$$

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

# Church Pairs

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

$$\text{pair} = [xyf] f x y$$

$$\text{fst} = [\rho] \rho K$$

$$\text{snd} = [\rho] \rho (KI)$$

We have:

$$\text{pair } s t = [xyf] f x y \rightarrow^* [f] f s t$$

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

# Church Pairs

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

$$\text{pair} = [xyf] f x y$$

$$\text{fst} = [\rho] \rho K$$

$$\text{snd} = [\rho] \rho (KI)$$

We have:

$$\text{pair } s t = [xyf] f x y \rightarrow^* [f] f s t$$

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

# Church Pairs

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

$$\text{pair} = [xyf] f x y$$

$$\text{fst} = [\rho] \rho K$$

$$\text{snd} = [\rho] \rho (KI)$$

We have:

$$\text{pair } s t = [xyf] f x y \rightarrow^* [f] f s t$$

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

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

# Church Pairs

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

$$\text{pair} = [xyf] f x y$$

$$\text{fst} = [\rho] \rho K$$

$$\text{snd} = [\rho] \rho (KI)$$

We have:

$$\text{pair } s t = [xyf] f x y \rightarrow^* [f] f s t$$

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

$$\text{snd } (\text{pair } s t) \rightarrow^* ([\rho] \rho (KI)) ([f] f s t)$$



# Church Pairs

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

$$\text{pair} = [xyf] f x y$$

$$\text{fst} = [\rho] \rho K$$

$$\text{snd} = [\rho] \rho (KI)$$

We have:

$$\text{pair } s t = [xyf] f x y \rightarrow^* [f] f s t$$

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

$$\text{snd } (\text{pair } s t) \rightarrow^* ([\rho] \rho (KI)) ([f] f s t) \rightarrow^* ([f] f s t)(KI)$$

# Church Pairs

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

$$\text{pair} = [xyf] f x y$$

$$\text{fst} = [\rho] \rho K$$

$$\text{snd} = [\rho] \rho (KI)$$

We have:

$$\text{pair } s t = [xyf] f x y \rightarrow^* [f] f s t$$

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

$$\text{snd } (\text{pair } s t) \rightarrow^* ([\rho] \rho (KI)) ([f] f s t) \rightarrow^* ([f] f s t)(KI) \rightarrow^* KI s t$$

# Church Pairs

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

$$\text{pair} = [xyf] f x y$$

$$\text{fst} = [\rho] \rho K$$

$$\text{snd} = [\rho] \rho (KI)$$

We have:

$$\text{pair } s t = [xyf] f x y \rightarrow^* [f] f s t$$

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

$$\text{snd } (\text{pair } s t) \rightarrow^* ([\rho] \rho (KI)) ([f] f s t) \rightarrow^* ([f] f s t)(KI) \rightarrow^* KI s t \rightarrow^* t$$

# Church Pairs

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

$$\text{pair} = [xyf] f x y$$

$$\text{fst} = [p] p K$$

$$\text{snd} = [p] p (KI)$$

We have:

$$\text{pair } s t = [xyf] f x y \rightarrow^* [f] f s t$$

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

$$\text{snd } (\text{pair } s t) \rightarrow^* ([p] p (KI)) ([f] f s t) \rightarrow^* ([f] f s t)(KI) \rightarrow^* KI s t \rightarrow^* t$$

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:  $\bar{n} = \lambda f. \lambda x. f^n(x)$ .

# Church Numerals

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

$$\bar{0} = KI$$

$$\overline{n+1} = ([nf\lambda] f (n f x)) \bar{n}$$

# Church Numerals

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

$$\bar{0} = KI$$

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

# Church Numerals

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

$$\bar{0} = KI$$

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

$$\bar{0} f x \rightarrow^* x$$



# Church Numerals

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

$$\bar{0} = KI$$

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

$$\bar{0} f x \rightarrow^* x$$

$$\overline{n+1} f x$$

# Church Numerals

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

$$\bar{0} = KI$$

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

$$\bar{0} f x \rightarrow^* x$$

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

# Church Numerals

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

$$\bar{0} = KI$$

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

$$\bar{0} f x \rightarrow^* x$$

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

# Church Numerals

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

$$\bar{0} = KI$$

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

$$\bar{0} f x \rightarrow^* x$$

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

# Church Numerals

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

$$\bar{0} = KI$$

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

$$\bar{0} f x \rightarrow^* x$$

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

# Church Numerals

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

$$\bar{0} = KI$$

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

$$\bar{0} f x \rightarrow^* x$$

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

# Church Numerals

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

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

$$\bar{0} f x \rightarrow^* x$$

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

# Church Numerals

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

$$\bar{0} = KI$$

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

$$\bar{0} f x \rightarrow^* x$$

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



# Church Numerals

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

$$\bar{0} = KI$$

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

$$\bar{0} f x \rightarrow^* x$$

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

## Church Numerals

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

$$\bar{0} = KI$$

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

$$\bar{0} f x \rightarrow^* x$$

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

## Church Numerals

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

$$\bar{0} = KI$$

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

$$\bar{0} f x \rightarrow^* x$$

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

# Computation with Church Numerals

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

# Computation with Church Numerals

- $\text{plus} = [m\ n\ f\ x]\ m\ f\ (n\ f\ x)$
- $\text{succ} = [n\ f\ x]\ f\ (n\ f\ x)$

# Computation with Church Numerals

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

# Computation with Church Numerals

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

# Computation with Church Numerals

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



# 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
- $\mu$ -recursive functions

# Computable Functions in Combinatory Logic

The class of  $\mu$ -recursive (found by Kleene) functions is build from:

- *zero*: 0

# Computable Functions in Combinatory Logic

The class of  $\mu$ -recursive (found by Kleene) functions is build from:

- *zero*: 0

in CL: *KI*

# Computable Functions in Combinatory Logic

The class of  $\mu$ -recursive (found by Kleene) functions is build from:

- *zero*: 0
- *successor*:  $S(n) = n+1$

in CL: *KI*

# Computable Functions in Combinatory Logic

The class of  $\mu$ -recursive (found by Kleene) functions is build from:

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

# Computable Functions in Combinatory Logic

The class of  $\mu$ -recursive (found by Kleene) functions is build from:

- *zero*: 0 in CL:  $KI$
- *successor*:  $S(n) = n+1$  in CL:  $[nf x] f (nf x)$
- *projection functions*:  $\Pi_k^i(n_1, \dots, n_k) = n_i$

# Computable Functions in Combinatory Logic

The class of  $\mu$ -recursive (found by Kleene) functions is build from:

- *zero*:  $0$  in CL:  $KI$
- *successor*:  $S(n) = n+1$  in CL:  $[nf x] f (nf x)$
- *projection functions*:  $\Pi_k^i(n_1, \dots, n_k) = n_i$  in CL:  $\text{pair}, \text{fst}, \text{snd}, \dots$



# Computable Functions in Combinatory Logic

The class of  $\mu$ -recursive (found by Kleene) functions is build from:

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

# Computable Functions in Combinatory Logic

The class of  $\mu$ -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*:  $\Pi_k^i(n_1, \dots, n_k) = n_i$  in CL:  $\text{pair}, \text{fst}, \text{snd}, \dots$
- *composition*:  $f(x_1, \dots, x_n) = g(h_1(\vec{x}), \dots, h_m(\vec{x}))$   
in CL:  $[\vec{x}]g(h_1 \vec{x}, \dots, h_m \vec{x})$

# Computable Functions in Combinatory Logic

The class of  $\mu$ -recursive (found by Kleene) functions is build from:

- *zero*:  $0$  in CL:  $KI$
- *successor*:  $S(n) = n+1$  in CL:  $[nf\ x] f(nf\ x)$
- *projection functions*:  $\Pi_k^i(n_1, \dots, n_k) = n_i$  in CL:  $\text{pair}, \text{fst}, \text{snd}, \dots$
- *composition*:  $f(x_1, \dots, x_n) = g(h_1(\vec{x}), \dots, h_m(\vec{x}))$   
in CL:  $[\vec{x}]g(h_1 \vec{x}, \dots, h_m \vec{x})$
- *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  $\mu$ -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*:  $\Pi_k^i(n_1, \dots, n_k) = n_i$  in CL:  $\text{pair}, \text{fst}, \text{snd}, \dots$
- *composition*:  $f(x_1, \dots, x_n) = g(h_1(\vec{x}), \dots, h_m(\vec{x}))$   
in CL:  $[\vec{x}]g(h_1 \vec{x}, \dots, h_m \vec{x})$
- *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  $f n \vec{x} = \text{isZero } n (g \vec{x}) (h (f (\text{pred } n) \vec{x}) n \vec{x})$

# Computable Functions in Combinatory Logic

The class of  $\mu$ -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*:  $\Pi_k^i(n_1, \dots, n_k) = n_i$  in CL:  $\text{pair}, \text{fst}, \text{snd}, \dots$
- *composition*:  $f(x_1, \dots, x_n) = g(h_1(\vec{x}), \dots, h_m(\vec{x}))$   
in CL:  $[\vec{x}]g(h_1 \vec{x}, \dots, h_m \vec{x})$
- *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  $f(u, \vec{x}) = 0$

# Computable Functions in Combinatory Logic

The class of  $\mu$ -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*:  $\Pi_k^i(n_1, \dots, n_k) = n_i$  in CL:  $\text{pair}, \text{fst}, \text{snd}, \dots$
- *composition*:  $f(x_1, \dots, x_n) = g(h_1(\vec{x}), \dots, h_m(\vec{x}))$   
in CL:  $[\vec{x}]g(h_1 \vec{x}, \dots, h_m \vec{x})$
- *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  $\mu u f \vec{x} = \text{isZero } (f u \vec{x}) u (\mu (\text{succ } u) f \vec{x})$

# Computable Functions in Combinatory Logic

The class of  $\mu$ -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*:  $\Pi_k^i(n_1, \dots, n_k) = n_i$  in CL:  $\text{pair}, \text{fst}, \text{snd}, \dots$
- *composition*:  $f(x_1, \dots, x_n) = g(h_1(\vec{x}), \dots, h_m(\vec{x}))$   
in CL:  $[\vec{x}]g(h_1 \vec{x}, \dots, h_m \vec{x})$
- *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.