Combinatory Logic / Simply-Typed Lambda Calculus

FOCS, Fall 2020

Combinators

A combinator is a lambda calculus expression that contains no free identifier

- All identifiers refer to parameters in enclosing abstractions

All encodings we have seen are combinators

<x y → x>
<x y → <s → (s x) y>>
<f x → f (f (f x))>

Combinatory logic

One of the simplest systems

Expressions:

- **I**, **K**, **S** are expressions
- M N is an expression if M, N are expressions

Simplification rules:

- ((**S** *a*) *b*) *c* = (*a c*) (*b c*)
- (**K** *a*) *b* = *a*
- **I** *a* = *a*

Combinatory logic and lambda calculus

You can implement combinatory logic in the lambda calculus:

S = <x y z → (x z) (y z)>

K = <x y → x>

I = <x → x>

((**S** *a*) *b*) *c* =

Combinatory logic and lambda calculus

You can implement combinatory logic in the lambda calculus:

S = <x y z → (x z) (y z)>

K = <x y → x>

I = <x → x>

((**S** *a*) *b*) *c* = ((<x y z → (x z) (y z)> *a*) *b*) *c*

Combinatory logic and lambda calculus

You can implement combinatory logic in the lambda calculus:

S = <x y z → (x z) (y z)>

K = <x y → x>

I = <x → x>

((**S** *a*) *b*) *c* = ((<x y z → (x z) (y z)> *a*) *b*) *c*
= (<y z → (*a* z) (y z)> *b*) *c*

Combinatory logic and lambda calculus

You can implement combinatory logic in the lambda calculus:

S = <x y z → (x z) (y z)>

K = <x y → x>

I = <x → x>

((**S** *a*) *b*) *c* = ((<x y z → (x z) (y z)> *a*) *b*) *c*
= (<y z → (*a* z) (y z)> *b*) *c*
= <z → (*a* z) (*b* z)> *c*

Combinatory logic and lambda calculus

You can implement combinatory logic in the lambda calculus:

S = <x y z → (x z) (y z)>

K = <x y → x>

I = <x → x>

((**S** *a*) *b*) *c* = ((<x y z → (x z) (y z)> *a*) *b*) *c*
= (<y z → (*a* z) (y z)> *b*) *c*
= <z → (*a* z) (*b* z)> *c*
= (*a* *c*) (*b* *c*)

Combinatory logic is Turing-complete

A computational model is Turing-complete if it can simulate Turing machines

The lambda calculus is Turing-complete:

- you can implement Q1 from Homework 4 in the lambda calculus
- you can implement it with combinators

Combinatory logic is Turing-complete:

- you can implement any closed lambda expressions

Translation algorithm

<x → x> ⇒ **I**

<x → M> ⇒ **K** M (if x not free in M)

<x → M N> ⇒ (**S** <x → M>) <x → N>

Repeatedly translate abstractions until none remain

Example

<x → <y → y x>> =

Example

$$<x \rightarrow <y \rightarrow y\ x>> \ = \ <x \rightarrow (\mathbf{S}\ <y \rightarrow y>)\ <y \rightarrow x>>$$

Example

<x → <y → y x>> = <x → (**S** <y → y>) <y → x>>

= <x → (**S I**) <y → x>>

Example

<x → <y → y x>> = <x → (**S** <y → y>) <y → x>>

= <x → (**S I**) <y → x>>

= <x → (**S I**) (**K** x)>

Example

<x → <y → y x>> = <x → (**S** <y → y>) <y → x>>

= <x → (**S** **I**) <y → x>>

= <x → (**S** **I**) (**K** x)>

= (**S** <x → **S** **I**>) <x → **K** x>

Example

$$
\begin{aligned}
<x \to <y \to y\ x>> &= <x \to (\mathbf{S}\ <y \to y>)\ <y \to x>> \\
&= <x \to (\mathbf{S}\ \mathbf{I})\ <y \to x>> \\
&= <x \to (\mathbf{S}\ \mathbf{I})\ (\mathbf{K}\ x)> \\
&= (\mathbf{S}\ <x \to \mathbf{S}\ \mathbf{I}>)\ <x \to \mathbf{K}\ x> \\
&= (\mathbf{S}\ (\mathbf{K}\ (\mathbf{S}\ \mathbf{I})))\ <x \to \mathbf{K}\ x>
\end{aligned}
$$

Example

$$
\begin{aligned}
<x \rightarrow <y \rightarrow y\ x>> &= <x \rightarrow (\mathbf{S}\ <y \rightarrow y>)\ <y \rightarrow x>> \\
&= <x \rightarrow (\mathbf{S}\ \mathbf{I})\ <y \rightarrow x>> \\
&= <x \rightarrow (\mathbf{S}\ \mathbf{I})\ (\mathbf{K}\ x)> \\
&= (\mathbf{S}\ <x \rightarrow \mathbf{S}\ \mathbf{I}>)\ <x \rightarrow \mathbf{K}\ x> \\
&= (\mathbf{S}\ (\mathbf{K}\ (\mathbf{S}\ \mathbf{I})))\ <x \rightarrow \mathbf{K}\ x> \\
&= (\mathbf{S}\ (\mathbf{K}\ (\mathbf{S}\ \mathbf{I})))\ ((\mathbf{S}\ <x \rightarrow \mathbf{K}>)\ <x \rightarrow x>)
\end{aligned}
$$

Example

<x → <y → y x>> = <x → (**S** <y → y>) <y → x>>

= <x → (**S I**) <y → x>>

= <x → (**S I**) (**K** x)>

= (**S** <x → **S I**>) <x → **K** x>

= (**S** (**K** (**S I**))) <x → **K** x>

= (**S** (**K** (**S I**))) ((**S** <x → **K**>) <x → x>)

= (**S** (**K** (**S I**))) ((**S** (**K K**)) <x → x>)

Example

<x → <y → y x>> = <x → (**S** <y → y>) <y → x>>
= <x → (**S I**) <y → x>>
= <x → (**S I**) (**K** x)>
= (**S** <x → **S I**>) <x → **K** x>
= (**S** (**K** (**S I**))) <x → **K** x>
= (**S** (**K** (**S I**))) ((**S** <x → **K**>) <x → x>)
= (**S** (**K** (**S I**))) ((**S** (**K K**)) <x → x>)
= (**S** (**K** (**S I**))) ((**S** (**K K**)) **I**)

Validation: (((**S** (**K** (**S I**))) ((**S** (**K K**)) **I**)) *a*) *b* = *b a*

Example

(((**S** (**K** (**S** **I**))) ((**S** (**K** **K**)) **I**)) *a*) *b*

Example

(((**S** (**K** (**S** **I**))) ((**S** (**K** **K**)) **I**)) *a*) *b*

= (((**K** (**S** **I**)) *a*) (((**S** (**K** **K**)) **I**) *a*)) *b*

Example

(((**S** (**K** (**S** **I**))) ((**S** (**K** **K**)) **I**)) *a*) *b*

= (((**K** (**S** **I**)) *a*) (((**S** (**K** **K**)) **I**) *a*)) *b*

= ((**S** **I**) (((**S** (**K** **K**)) **I**) *a*)) *b*

Example

(((**S** (**K** (**S** **I**))) ((**S** (**K** **K**)) **I**)) *a*) *b*

= (((**K** (**S** **I**)) *a*) (((**S** (**K** **K**)) **I**) *a*)) *b*

= ((**S** **I**) (((**S** (**K** **K**)) **I**) *a*)) *b*

= (**I** b) ((((**S** (**K** **K**)) **I**) *a*) *b*)

Example

(((**S** (**K** (**S** **I**))) ((**S** (**K** **K**)) **I**)) *a*) *b*

= (((**K** (**S** **I**)) *a*) (((**S** (**K** **K**)) **I**) *a*)) *b*

= ((**S** **I**) (((**S** (**K** **K**)) **I**) *a*)) *b*

= (**I** b) ((((**S** (**K** **K**)) **I**) *a*) *b*)

= *b* ((((**S** (**K** **K**)) **I**) *a*) *b*)

Example

$$(((\mathbf{S}\ (\mathbf{K}\ (\mathbf{S}\ \mathbf{I})))\ ((\mathbf{S}\ (\mathbf{K}\ \mathbf{K}))\ \mathbf{I}))\ a)\ b$$

$$= (((\mathbf{K}\ (\mathbf{S}\ \mathbf{I}))\ a)\ (((\mathbf{S}\ (\mathbf{K}\ \mathbf{K}))\ \mathbf{I})\ a))\ b$$

$$= ((\mathbf{S}\ \mathbf{I})\ (((\mathbf{S}\ (\mathbf{K}\ \mathbf{K}))\ \mathbf{I})\ a))\ b$$

$$= (\mathbf{I}\ \mathrm{b})\ ((((\mathbf{S}\ (\mathbf{K}\ \mathbf{K}))\ \mathbf{I})\ a)\ b)$$

$$= b\ ((((\mathbf{S}\ (\mathbf{K}\ \mathbf{K}))\ \mathbf{I})\ a)\ b)$$

$$= b\ ((((\mathbf{K}\ \mathbf{K})\ a)\ (\mathbf{I}\ a))\ b)$$

Example

$(((\mathbf{S}\ (\mathbf{K}\ (\mathbf{S}\ \mathbf{I})))\ ((\mathbf{S}\ (\mathbf{K}\ \mathbf{K}))\ \mathbf{I}))\ a)\ b$

$\quad = (((\mathbf{K}\ (\mathbf{S}\ \mathbf{I}))\ a)\ (((\mathbf{S}\ (\mathbf{K}\ \mathbf{K}))\ \mathbf{I})\ a))\ b$

$\quad = ((\mathbf{S}\ \mathbf{I})\ (((\mathbf{S}\ (\mathbf{K}\ \mathbf{K}))\ \mathbf{I})\ a))\ b$

$\quad = (\mathbf{I}\ \mathrm{b})\ ((((\mathbf{S}\ (\mathbf{K}\ \mathbf{K}))\ \mathbf{I})\ a)\ b)$

$\quad = b\ ((((\mathbf{S}\ (\mathbf{K}\ \mathbf{K}))\ \mathbf{I})\ a)\ b)$

$\quad = b\ ((((\mathbf{K}\ \mathbf{K})\ a)\ (\mathbf{I}\ a))\ b)$

$\quad = b\ ((\mathbf{K}\ (\mathbf{I}\ a))\ b)$

Example

$$
\begin{aligned}
&(((\mathbf{S}\ (\mathbf{K}\ (\mathbf{S}\ \mathbf{I})))\ ((\mathbf{S}\ (\mathbf{K}\ \mathbf{K}))\ \mathbf{I}))\ a)\ b \\
&\qquad = (((\mathbf{K}\ (\mathbf{S}\ \mathbf{I}))\ a)\ (((\mathbf{S}\ (\mathbf{K}\ \mathbf{K}))\ \mathbf{I})\ a))\ b \\
&\qquad = ((\mathbf{S}\ \mathbf{I})\ (((\mathbf{S}\ (\mathbf{K}\ \mathbf{K}))\ \mathbf{I})\ a))\ b \\
&\qquad = (\mathbf{I}\ \mathrm{b})\ ((((\mathbf{S}\ (\mathbf{K}\ \mathbf{K}))\ \mathbf{I})\ a)\ b) \\
&\qquad = b\ ((((\mathbf{S}\ (\mathbf{K}\ \mathbf{K}))\ \mathbf{I})\ a)\ b) \\
&\qquad = b\ ((((\mathbf{K}\ \mathbf{K})\ a)\ (\mathbf{I}\ a))\ b) \\
&\qquad = b\ ((\mathbf{K}\ (\mathbf{I}\ a))\ b) \\
&\qquad = b\ (\mathbf{I}\ a)
\end{aligned}
$$

Example

$$
\begin{aligned}
&(((\mathbf{S}\ (\mathbf{K}\ (\mathbf{S}\ \mathbf{I})))\ ((\mathbf{S}\ (\mathbf{K}\ \mathbf{K}))\ \mathbf{I}))\ a)\ b \\
&\qquad = (((\mathbf{K}\ (\mathbf{S}\ \mathbf{I}))\ a)\ (((\mathbf{S}\ (\mathbf{K}\ \mathbf{K}))\ \mathbf{I})\ a))\ b \\
&\qquad = ((\mathbf{S}\ \mathbf{I})\ (((\mathbf{S}\ (\mathbf{K}\ \mathbf{K}))\ \mathbf{I})\ a))\ b \\
&\qquad = (\mathbf{I}\ \mathrm{b})\ ((((\mathbf{S}\ (\mathbf{K}\ \mathbf{K}))\ \mathbf{I})\ a)\ b) \\
&\qquad = b\ ((((\mathbf{S}\ (\mathbf{K}\ \mathbf{K}))\ \mathbf{I})\ a)\ b) \\
&\qquad = b\ ((((\mathbf{K}\ \mathbf{K})\ a)\ (\mathbf{I}\ a))\ b) \\
&\qquad = b\ ((\mathbf{K}\ (\mathbf{I}\ a))\ b) \\
&\qquad = b\ (\mathbf{I}\ a) \\
&\qquad = b\ a
\end{aligned}
$$

Simply-typed lambda calculus

A model for understanding modern programming languages

1) Start with the lambda calculus

2) Add integers + some interesting constants

   0, 1, 2, 3, …, -1, -2, -3, **…** are expressions
   succ is an expression

3) Add special specific simplifications rules for constants

   succ i = i + 1

Simply-typed lambda calculus

Expressions M, N:

x, y, z, ...
0, 1, 2, 3, ... , -1, -2, -3, ...
succ
<x : T → M> (for an expression M)
M N (for expressions M, N)

Types T, U:

$\mathbb{Z}$
T → U (for types T, U)

Examples

succ2 = $<x : \mathbb{Z} \to \text{succ (succ } x)>$

pair = $<x : \mathbb{Z} \to <y : \mathbb{Z} \to <s : (\mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z})) \to (s\ x)\ y>>>$

first = $<p : ((\mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z})) \to \mathbb{Z}) \to p <x : \mathbb{Z} \to <y : \mathbb{Z} \to x>>>$

Intuitively:

succ2 has type $\mathbb{Z} \to \mathbb{Z}$
pair has type $\mathbb{Z} \to (\mathbb{Z} \to ((\mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z})) \to \mathbb{Z}))$
(**pair** 10) 20 has type $(\mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z})) \to \mathbb{Z}$
first has type $((\mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z})) \to \mathbb{Z}) \to \mathbb{Z}$

Examples

succ2 = <x : ℤ → succ (succ x)>

pair = <x : ℤ → <y : ℤ → <s : (ℤ → (ℤ → ℤ)) → (s x) y>>>

first = <p : ((ℤ → (ℤ → ℤ)) → ℤ) → p <x : ℤ → <y : ℤ → x>>>

Sample simplifications:

succ2 (**first** ((**pair** 10) 20) = … = 12

succ2 ((**pair** 10) 20) = … = succ (succ <s : (ℤ → (ℤ → ℤ)) → (s 10) 20>)

Type-checking algorithm

Algorithm to get the type of an expression in a context C

Context = assignment of types to identifiers

- *i* has type $\mathbb{Z}$ in any context
- succ has type $\mathbb{Z} \to \mathbb{Z}$ in any context
- x has type T in context C with $(x:T) \in C$
- <x : T → M> has type T → U in context C if:
  M has type U in context C U {x:T}
- M N has type U in context C if:
  M has type T → U in context C and N has type T in context C

When I don't mention a context, it's the empty context

Example

succ (succ 10) has type $\mathbb{Z}$ (in context $\varnothing$)

Example

succ (succ 10) has type $\mathbb{Z}$ (in context $\varnothing$)

  succ has type $\mathbb{Z} \longrightarrow \mathbb{Z}$

  succ 10 has type $\mathbb{Z}$

Example

succ (succ 10) has type $\mathbb{Z}$ (in context $\varnothing$)

   succ has type $\mathbb{Z} \to \mathbb{Z}$

   succ 10 has type $\mathbb{Z}$

      succ has type $\mathbb{Z} \to \mathbb{Z}$

      10 has type $\mathbb{Z}$

Example

succc2 = <x : $\mathbb{Z} \rightarrow$ succ (succ x)> has type $\mathbb{Z} \rightarrow \mathbb{Z}$

Example

succ2 = <x : $\mathbb{Z} \to$ succ (succ x)> has type $\mathbb{Z} \to \mathbb{Z}$

succ (succ x) has type $\mathbb{Z}$ in context $\{x:\mathbb{Z}\}$

Example

succ2 = <x : $\mathbb{Z} \rightarrow$ succ (succ x)> has type $\mathbb{Z} \rightarrow \mathbb{Z}$

- succ (succ x) has type $\mathbb{Z}$ in context $\{x:\mathbb{Z}\}$
  - succ has type $\mathbb{Z} \rightarrow \mathbb{Z}$ in context $\{x:\mathbb{Z}\}$
  - succ x has type $\mathbb{Z}$ in context $\{x:\mathbb{Z}\}$

Example

succ2 = <x : ℤ → succ (succ x)> has type ℤ → ℤ

succ (succ x) has type ℤ in context {x:ℤ}

succ has type ℤ → ℤ in context {x:ℤ}

succ x has type ℤ in context {x:ℤ}

succ has type ℤ → ℤ in context {x:ℤ}

x has type ℤ in context {x:ℤ}

Example

pair = $<x : \mathbb{Z} \to <y : \mathbb{Z} \to <s : (\mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z})) \to (s\ x)\ y>>>$ has type $\mathbb{Z} \to (\mathbb{Z} \to ((\mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z})) \to \mathbb{Z}))$

$<y : \mathbb{Z} \to <s : (\mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z})) \to (s\ x)\ y>>>$ has type $\mathbb{Z} \to ((\mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z})) \to \mathbb{Z})$ in context $\{x{:}\mathbb{Z}\}$

$<s : (\mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z})) \to (s\ x)\ y>$ has type $(\mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z})) \to \mathbb{Z}$ in context $\{x{:}\mathbb{Z},\ y{:}\mathbb{Z}\}$

$(s\ x)\ y$ has type $\mathbb{Z}$ in context $\{x{:}\mathbb{Z},\ y{:}\mathbb{Z},\ s{:}(\mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z}))\}$

$(s\ x)$ has type $\mathbb{Z} \to \mathbb{Z}$ in context $\{x{:}\mathbb{Z},\ y{:}\mathbb{Z},\ s{:}(\mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z}))\}$

s has type $\mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z})$ in context $\{x{:}\mathbb{Z},\ y{:}\mathbb{Z},\ s{:}(\mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z}))\}$

x has type $\mathbb{Z}$ in context $\{x{:}\mathbb{Z},\ y{:}\mathbb{Z},\ s{:}(\mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z}))\}$

y has type $\mathbb{Z}$ in context $\{x{:}\mathbb{Z},\ y{:}\mathbb{Z},\ s{:}(\mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z}))\}$

Example

first = <p : ((ℤ → (ℤ → ℤ)) → ℤ) → p <x : ℤ → <y : ℤ → x>>> has type ((ℤ → (ℤ → ℤ)) → ℤ) → ℤ

- p <x : ℤ → <y : ℤ → x>> has type ℤ in context {p:((ℤ → (ℤ → ℤ)) → ℤ)}
  - p has type (ℤ → (ℤ → ℤ)) → ℤ in context {p:((ℤ → (ℤ → ℤ)) → ℤ)}
  - <x : ℤ → <y : ℤ → x>> has type ℤ → (ℤ → ℤ) in context {p:((ℤ → (ℤ → ℤ)) → ℤ)}
    - <y : ℤ → x> has type ℤ → ℤ in context {p:((ℤ → (ℤ → ℤ)), x:ℤ}
      - x has type ℤ in context {p:((ℤ → (ℤ → ℤ)), x:ℤ, y:ℤ}

Example

((pair 10) 20) has type $(\mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z})) \to \mathbb{Z}$

pair 10 has type $\mathbb{Z} \to ((\mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z})) \to \mathbb{Z})$

pair has type $\mathbb{Z} \to (\mathbb{Z} \to ((\mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z})) \to \mathbb{Z}))$

10 has type $\mathbb{Z}$

20 has type $\mathbb{Z}$

Example

succ2 (first ((pair 10) 20)) has type $\mathbb{Z}$

succ2 has type $\mathbb{Z} \to \mathbb{Z}$

first ((pair 10) 20) has type $\mathbb{Z}$

first has type $((\mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z})) \to \mathbb{Z}) \to \mathbb{Z}$

(pair 10) 20 has type $(\mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z})) \to \mathbb{Z}$

succ2 ((pair 10) 20) doesn't have a type because:

succ2 has type $\mathbb{Z} \to \mathbb{Z}$

(pair 10) 20 has type $(\mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z})) \to \mathbb{Z}$

Type soundness

A value is an expression of the form

i

succ

<x : T → M>

Theorem: if M has type T, then there exists a value N of type T with M=N

In English: if M has type T, then you can simplify M to a value