Types

Lambda Terms

- Variables: x y z…
- Abstractions (function creation): λx.M
  - λx.M: x ↦ M[x]
  - parameter x; body M
- Applications: MN
  - meaning M(N)

Currying

- Unary functions suffice
- Instead of M(X,Y) use M(X)(Y)
    - Applying M to X and then applying result to Y
    - Often written as MXY
    - Understood as (MX)Y
    - +13 means (+1)3, where +1 increments any number

Beta

Apply an abstraction to a term

- (λx.M[x,x,…,x])N ➨ M[N,N,…,N]
    - replace all <u>free</u> occurrences of x in M with N

Combinatory Logic

$$Sxyz = (xz)(yz)$$

$$Kxy = x$$

$$Ix = x$$

Combinatory Rewriting

Sxyz ➨ (xz)(yz)

Kxy ➨ x

Ix ➨ x

I Combinator

Sxyz ➨ (xz)(yz)

Kxy ➨ x

- (SKK)x ➨ (Kx)(Kx) ➨ x
- Let I = SKK

Y Combinator

Yz = z(Yz)

S(K(SII))(S(S(KS)K)(K(SII)))

Lemma: SIIx = xx

S (K(SII)) (S(S(KS)K)(K(SII))) z

(K(SII))z (S (S(KS)K) (K(SII)) z)

SII (S(KS)Kz (K(SII) z)

SII ((KS)z(Kz) (SII))

SII (S(Kz)(SII))

…

Base Types

- Integers
- Booleans
- Characters
- Floating point

Polymorphic Types

- Lists (of anything)
- Stacks
- Trees
- ….

Function Types

- Program $N \to N$
- Interpreter $(N \to N) \times N \to N$
- Compiler $(N \to N) \to (A \to A)$

Arrow Types

- Notation
  - t : τ (term t has type τ)
- Suppose x : σ and t : τ
  - λx.t : $\sigma \rightarrow \tau$
- Suppose s : $\tau \rightarrow \sigma$ and t : τ
  - st : σ

Nontermination

$(\lambda x.xx)(\lambda x.xx)$ rewrites to itself

- $(\lambda x.xx)(\lambda x.xx) \Rightarrow (\lambda x.xx)(\lambda x.xx)$

Nontermination

What kind of function may be applied to itself?

- interpreter
- partial evaluator
- compiler
- compiler-compiler
- compiler-compiler-compiler

Well-Typed Terms

- Lambda terms
  - Λ
- Some terms can be typed
  - Λ
- Some cannot

  $\lambda x.xx$

Well-Typed Terms

- Have normal forms
  - Easy (Turing)
- Have no immortal (nonterminating) reductions
  - Hard (Tait)

Termination Properties

- s is terminating iff all t, such s ➨ t, are terminating
- If (st) is terminating, then s and t are
- If t is terminating, then (xt) is
- If s and t[s] are terminating, then $(\lambda x.t)s$ is

Computability

- A term of <u>base</u> type is computable iff it is terminating.

- A term of <u>arrow</u> type is computable if applying it to a computable term always gives a computable term.

Lemmata

- 1: If t is computable, then it is terminating.
- 2: If s[t] is computable and t is terminating, then (λx.s)t is computable.
- 3: If substitution α is computable, then so is sα.

Theorem

- Every (typeable) term is computable, hence, terminating.
- Proof: Empty α.

Lemmata

- 1: If t is computable, then it is terminating.
    - By induction on type structure.
- 2: If s[t] is computable …, then (λx.s)t is.
    - By induction on type structure.
- 3: If substitution α is computable, then so is sα.
    - By induction on term structure.

Lemma 1

- a: If s,…,t are terminating, then w=xs…t is computable.
- b: If w is computable, then it is terminating.

Lemma 1: Base

- a: If s,…,t are terminating, then w=xs…t is computable.
- b: If w is computable, then it is terminating.
- w : base type
  - a: xs…t is terminating, hence computable
  - b: by definition

Lemma 1: Arrow

- a: If s,…,t are terminating, then w=xs…t is computable.
- b: If w is computable, then it is terminating.
- w : $\sigma \to \tau$
    - a: xs…tu : τ is computable by induction
    - b: By def. wv : τ is computable for computable v : σ. By ind. wv terminating; so w is.

Lemma 2

- If s[t] is computable and t is terminating, then (λx.s[x])t is computable.

Lemma 2

- Given: s[t] is computable, t terminating.
  - By L1b, s[t] is terminating.
  - Hence s[x] is also terminating.

Lemma 2

- Consider any computable $u_1,\ldots,u_n$ (of appropriate type) such that $(\lambda x.s[x])tu_1\ldots u_n$ is basic ($n \geq 0$).
- We need to show $(\lambda x.s[x])tu_1\ldots u_n$ terminating, hence computable (by def.).
- Computability of each prefix $(\lambda x.s[x])tu_1\ldots u_i$ will follow.

Lemma 2

- We need to show $(\lambda x.s[x])tu_1 \ldots u_n$ terminating.
- $s[t]$ is computable; so $s[t]u_1 \ldots u_n$ is also (by def.) computable and terminating.
- $(\lambda x.s[x])tu_1 \ldots u_n$ ➨… ➨ $(\lambda x.s'[x])t'u'_1 \ldots u'_n$ ➨ $s'[t']u'_1 \ldots u'_n$ which is terminating, since $s[t]u_1 \ldots u_n$ is.

Lemma 3

- If substitution α is computable (all the terms to which variables map are computable), then so is $s\alpha$.

Lemma 3

- If α is computable, then so is sα.
  - xα is either the variable x or a computable term t
  - (uv)α = (uα)(vα) both parts of which are computable by induction, and so (uα)(vα) is by def.
  - Let s=(λx.t). Then sα=(λx.tα'), where α' is α <u>without</u> any substitution for x. Consider any computable u. By ind. tα'[x↦u] is computable. By L2, (sα)u=(λx.tα')u is computable. So, by def. sα is.

Two Dimensions

	X	XX	LX	L XX	X LX	XXX	LX X
o							
o-o							
o-(o-o)							
(o-o)-o							

Two Dimensions

	X	XX	LX	L XX	X LX	XXX	LX X
o							
o-o							
o-(o-o)							
(o-o)-o							

Two Dimensions

	X	XX	LX	L XX	X LX	XXX	LX X
o							
o-o							
o-(o-o)							
(o-o)-o							